Computer Shop

Karl-Heinz Koch

ATARI Spiele programmieren

Schritt für Schritt

Springer Basel AG

B

Computer Shop
Band 21

Karl-Heinz Koch

ATARI Spiele programmieren

Schritt für Schritt

Springer Basel AG

ISBN 978-3-7643-1659-4 ISBN 978-3-0348-6435-0 (eBook)
DOI 10.1007/978-3-0348-6435-0

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Koch, Karl-Heinz
ATARI, Spiele programmieren : Schritt für Schritt /
Karl-Heinz Koch. – Basel ; Boston ; Stuttgart :
Birkhäuser, 1984.
(Computer-Shop ; Bd. 21)
ISBN 978-3-7643-1659-4
NE: GT

Die vorliegende Publikation ist urheberrechtlich geschützt.
Alle Rechte vorbehalten. Kein Teil dieses Buches darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form durch Fotokopie, Mikrofilm, Kassetten oder andere Verfahren reproduziert werden. Auch die Rechte der Wiedergabe durch Vortrag, Funk und Fernsehen bleiben vorbehalten.

© Springer Basel AG 1984
Ursprünglich erschienen bei Birkhäuser Verlag, Basel 1984
Umschlaggestaltung: Bruckmann & Partner

Inhalt

0	Was?	7
1	BASIC im Laufschritt	9
1.1	Hübsch der Reihe nach	11
1.2	Variablen-Varianten	14
1.2.1	LET it be	16
1.2.2	Puppet on a STRING	18
1.2.3	Andere DIMensionen	20
1.3	Ein paar Brocken BASIC	23
1.3.1	PRINTen mit ?	24
1.3.2	I. wie INPUT	25
1.3.3	O gott, Otto: GOTO	28
1.3.4	GOSUBkultur	30
1.3.5	Wenn IF dann THEN	32
1.3.6	FOR the NEXT STEP: Schleifen	33
1.4	Ab hier geht's bunt zu	35
1.4.1	GRAPHICS-GR.affiti	36
1.4.2	MultiCOLOR	37
1.4.3	Farbenrausch	39
1.4.4	Zeichensprache	41
1.4.5	POS.itionen	43
1.4.6	Komm, PLOT auf den Bildschirm	47
1.4.7	Fill-Horn	50
1.5	Hilfe!	52
2	Bildschirmbilder	57
2.1.1	Lange Leitung	58
2.1.2	Mattscheibe	59
2.1.3	Schöner Schein	61
2.2	Modell mit 16 Gängen	64
2.2.0	GRAPHICS 0	68
2.2.1	GRAPHICS 1 / 17	72
2.2.2	GRAPHICS 2 / 18	75
2.2.3	GRAPHICS 3 / 19	79
2.2.4	GRAPHICS 4 / 20	80
2.2.5	GRAPHICS 5 / 21	81
2.2.6	GRAPHICS 6 / 22	82
2.2.7	GRAPHICS 7 / 23	83
2.2.8	GRAPHICS 8 / 24	84

2.2.9	GRAPHICS 9	86
2.2.10	GRAPHICS 10	88
2.2.11	GRAPHICS 11	90
2.2.12	GRAPHICS 12 / 28	92
2.2.13	GRAPHICS 13 / 29	94
2.2.14	GRAPHICS 14 / 30	96
2.2.15	GRAPHICS 15 / 31	97
3	Wie die Bilder laufen lernen	98
3.1	Druck-Grafik	99
3.2	Zufallslösung	102
3.3	Berechnend	122
3.4	Scheinlösung	132
3.5	Fliegender Wechsel	141
4	Spiele!	154
4.1	Der Spieler	156
4.1.1	Über STICK und Stein	157
4.1.2	Bewegte Zeichen	163
4.1.3	Genossen und Geschosse	170
4.2	Das Spielfeld	182
4.3	Die Feinde	190
4.3.1	Auf ausgetretenen Pfaden	191
4.3.2	Schicksalsschläge	194
4.3.3	Gnadenlose Päzision	197
4.4	Zeichen der Zeit	200
4.4.1	Zeitgerecht	201
4.4.2	Zeitdruck	204
4.5	Wertsachen	206
4.6	Der gute Ton	212
4.6.1	Klang-Daten	215
4.6.2	Klang-Bilder	218
5	Appendix	220
5.1	Zahlenraten	221
5.2	Malpinsel	222
5.3	Farben-Schlüssel	224
5.4	Bit-Muster	227
5.5	GRAPHICS 1 und 2: COLOR n	231
5.6	ATASCII-Zeichen und interner Code	232
5.7	BASIC-Wörter mit Seitenreferenz	234

0 Was?

Bücher im Zeitalter der Computer? Ist das nicht wie eine Armbanduhr in einem antiken Drama! Oder ist es etwa umgekehrt: Der Computer braucht das Buch.

Jeder, der mit einem Denkzeug arbeiten will (oder muß?) und höhere Erwartungen an sich stellt als Knöpfchendrücken und Diskettenwechseln, braucht Bücher, in denen er nachschlagen kann. Was man alles über einen Computer, und zwar über das jeweils einzelne Gerät, wissen muß, kann niemand im Kopf haben. Tabellen, Listen, Befehle, Adressen, Wert: endlose Mengen von Zahlen, Zahlen, Zahlen; dezimal, dual, hexadezimal und noch einmal.

Wenn Sie sich (nach langem Hin und Her) endlich auch einen Elektronenrechner gekauft haben, dann stehen Sie mit Ihrer Neuerwerbung in der Regel ganz allein da. Was die Hersteller ihren Geräten als Anleitungsbroschüre beilegen, kann meist nur mit viel Humor 'Einführung' genannt werden.

Dieses Buch ist eine Einführung. Was Sie als freier Bürger eines freien Landes damit anfangen, das bleibt ganz allein Ihnen überlassen, und ich will mich da gar nicht einmischen. Nur ein paar Vorschläge vielleicht?

Der einfachste Weg, mit diesem Werk fertig zu werden, ist, es auf den Müll zu werfen. Vielleicht fischt es jemand wieder heraus, der mehr damit anzufangen weiß.

Sie können das Buch natürlich auch ungelesen ins Regal stellen. Um einen nachhaltigen Eindruck bei Besuchern zu hinterlassen, muß man die Bücher schließlich nicht lesen, die man sich meterweise ins Regal stellt.

Als fortgeschrittener Computer-Mönch werden Sie womöglich die abgedruckten Programme abtippen. Es ist ja eine mönchische Tradition, abzuschreiben. Warum immer nur die Bibel? Etwa zwanzig Programme erwarten Sie, die kürzeren Beispiele nicht mitgerechnet.

Aber geschrieben habe ich dieses Buch eigentlich, damit Sie lernen, selbst Spiele zu programmieren. Aber dazu müßten Sie das Buch lesen (so richtig Seite für Seite) und die Beispiele ausprobieren (das geht bekanntlich noch vor studie-

ren) und die Erklärungen nachvollziehen - Schritt für Schritt.

Und so ist das Buch aufgebaut:

Im Kapitel 1 werden die grundlegenden BASIC- (falls Sie noch nicht wissen, was das nun wieder ist, haben Sie bitte noch ein klein wenig Geduld) Befehle wiederholt. Auch wenn Sie schon ein wenig programmieren können, sollten Sie diesen Abschnitt überfliegen, denn BASIC ist nicht gleich BASIC, und es werden auch einige Kommandos vorgestellt, die es in dieser Form nur bei ATARI gibt.

Die vielen Hersteller-Köche haben den BASIC-Brei nämlich schon ganz schön verdorben. Das Dutzend Grundbefehle ist wohl immer gleich (oder sollte ich vorsichtig sein und sagen - ähnlich), aber jedes Gerät ist mit irgendwelchen Besonderheiten ausgestattet, für die ebensolche besonderen Befehle kreiert wurden. ATARI ist stark in Farbe und Grafik und hat hier der Konkurrenz ein paar Kommandos voraus.

Im Kapitel 2 geht es dann so richtig ab. Hier erfahren Sie über Video-Technik und die verschiedenen Grafik-Betriebsarten des ATARI alles, was Sie fürs erste wissen sollten und zwischen die Seiten dieses Buches zu pressen war. Alles was ich zu sagen habe und vorprogrammiere, läuft auf den alten Modellen (600 und 800) genauso gut wie auf den Geräten der neuen XL-Serie. Wo es zwischen den beiden Generationen Unterschiede gibt, wird es erwähnt.

Kapitel 3: So viel Wissen darf nicht ungenutzt dahindämmern. Hier kommt Bewegung in die Grafik, und in

Kapitel 4 spielt sich dann schon einiges ab. Grundsätzliche Prinzipien von Computer-Spielen werden erörtert und Programmlösungen dafür vorgeschlagen. Wie kann das Spielfeld aussehen? Woher nehme ich die Spielfiguren? Und wie mache ich ihr Beine? Hilfe, Feinde! Aber wo kommen sie her? Wie die Widersacher ins Spiel kommen und wie man der dummen Maschine sinnvolle Spielzüge beibringt. Dann widmen wir uns einen Moment der Zeit, und auch die Wertung hat ihren Wert. Zu guter letzt gehts dann geräuschvoll zu: Haste Töne!

Ein Unterhaltungsprogramm in Kapitel 5 bietet eine handvoll Unterhaltungsprogramme als Vergnügen, das ja bekanntlich nach der Arbeit zu kommen hat. Im wesentlichen entspringen diese Grafik- und Spiel-Beispiele den Kenntnissen der vorangegangenen Kapitel. Jedes Listing ist ausführlich erklärt, und nebenbei werden noch ein paar Tips und Tricks verraten.

I BASIC im Laufschritt

BASIC ist eine Computer-Sprache. (Können Computer sprechen - oder gar verstehen?) Wie eine menschliche Sprache besteht BASIC aus einer Menge von Wörtern und einer Grammatik, die vorschreibt, wie die Wörter zu setzen sind.

Die BASIC-Wörter sind Befehle an die Maschine. Wenn ich meinem Toaster befehlen will, daß er heizt, drücke ich eine Taste. Bei einem elektronischen Rechner ist das nicht anders, nur daß er so viele Funktionen besitzt, daß er nicht für jede einzelne eine Taste hat. (Obwohl es das gibt. Die kleinen Sinclair-Computer ZX 81 und Spectrum verwenden für jeden BASIC-Befehl eine andere Taste der Tatatur.) Und weil für die Text- und Zifferneingaben ohnehin eine alpha-numerische Tastatur (Insider erkannt man daran, daß sie Keyboard sagen) benötigt wird, lag es nahe, die Befehle in Form von Wörtern einer reduzierten Sprache einzugeben.

Die Computer-Sprache vermittelt zwischen unserer Alltagssprache (damit ist selbstverständlich Englisch bzw. Amerikanisch gemeint, was sonst) und der Maschinensprache. Heute weiß ja Fritzchen schon, daß die künstliche Intelligenz der Computer über "0" und "1" nicht hinauskommt, "Aus" und "Ein", zwei Zustände eines Schalters, mehr nicht.

BASIC ist nichts anderes als eine Art Computer-Kindersprache, die jeder normale (= Englisch sprechende Mensch) sofort versteht. Schon mit ein paar Vokabeln und ein bißchen Grammatik lassen sich kleine Programme schreiben.

Aber wenn man anfängt, BASIC zu verstehen, ist man auch schon nicht mehr so richtig damit zufrieden, denn in BASIC geschriebenen Programme laufen relativ langsam und nutzen die Möglichkeiten, die das Denkzeug bietet, nicht richtig aus, weil sich mit dem BASIC-Wortschatz manches nicht sagen läßt. Trotzdem ist das Anfänger-allzweck-zeichenhafte-Anweisungssystem (Beginners All-purpose Symbolic Instruction Code) die zur Zeit am weitesten verbreitete Computer-Sprache, weil sie so herrlich leicht zu erlernen ist.

Spätestens wenn Sie BASIC auf dem einen Computer gelernt haben und dann einen anderen Typ betasten, merken Sie, daß wir in einer pluralistischen Gesellschaft leben. Zwar

sind die Grundbefehle bei den verschiedenen Fabrikaten so ähnlich, daß man sich rasch umgewöhnt. Drumherum jedoch sind so viele gerätespezifische Abweichungen, Besonderheiten, Beschränkungen und Erweiterungen, daß man praktisch für jeden Rechner oder zumindest für jede Marke eine eigene Sprache lernen muß. Weils so fein klingt, spricht man von BASIC-Dialekten.

In diesem Buch wird ATARI-BASIC gesprochen.

1.1 Hübsch der Reihe nach

Die Befehle (die Sie noch gar nicht kennen) werden über die Tastatur eingetippt. Alles was Sie eintippen, erscheint auf dem Bildschirm/Monitor, wenn er eingeschaltet ist. Dem Computer ist das egal. Der funktioniert auch, wenn Sie es nicht sehen.

Sie brauchen übrigens keine Angst zu haben, durch Herumprobieren irgendetwas kaputt zu machen. Das ist völlig unmöglich. Schlimmstenfalls stürzt der Rechner ab, das heißt er reagiert auf Eingaben nicht mehr. Dann können Sie ihn nur aus- und wieder einschalten.

Sie können so lange Befehle eintippen, bis der ganze Bildschirm voll ist und noch weiter, und es geschieht rein gar nichts. Die Maschine ist so einfältig, daß Sie ihr auch noch sagen müssen: "So, mein Guter, das ist meine Anweisung, also los!" Dafür gibt es die RETURN-Taste (auch Carriage Return). Sie gibt dem Computer das Kommando End Of Line, Ende der Zeile.

Tippen Sie jetzt FRE(0) und drücken die RETURN-Taste. Sofort verrät Ihnen ATARI, wieviel Speicherplat er im Moment noch für Sie übrig hat. Er druckt eine Zahl auf den Bildschirm, die Anzahl der freien Bytes.

Ein Bit (Binary digit) ist die Zähleinheit des Rechners: "0" oder "1". Acht Bit werden zu einem Byte zusammengefaßt. Der Speicher besteht aus einer Menge von Speicherzellen, die je ein Byte fassen können. Diese Speicherplätze sind fortlaufend durchnumeriert von 0 bis 65535, bei einem Computer mit 64 Kilobyte, denn ein kByte entspricht 1024 Byte.

Da ein nicht unerheblicher Teil des Speichers benötigt wird, um die Funktionen des Rechners zu steuern und noch einmal ein schönes Stück abgeht, um unser BASIC-Gestottere in die endlosen Reihen von Nullen und Einsen zu übersetzen, die allein von der Maschine verdaut werden können, ist der für den User (Anwender) verfügbare Speicher um einiges geringer.

Wenn der Wert an den Bildschirm ausgegeben ist, erscheint zwei Zeilen darunter READY. Damit sagt Ihnen der Computer: "Ich bin fertig. Was nun?"

Wenn Sie irgendetwas falsch machen, z.B. einen falschen Befehl eintippen oder den Befehl in Kleinbuchstaben schreiben oder vielleicht auch nur einen Tippfehler machen (das Elektronengehirn ist da sehr genau), erleben Sie, woran man sich als Programmierer beizeiten gewöhnen muß: ERROR-

Damit meint der Kasten, daß irgendwas nicht läuft und dann geht die Fehlersuche los. Wenn Sie Glück haben, steht noch eine Zahl dabei und Sie können im ATARI-Anleitungsbuch nachlesen, aus welcher Ecke das Problem weht.

Probieren Sie einmal:

GOTO 10 (und RETURN-Taste nicht vergessen). Na?

Mit ERROR- 12 meint die Rechenmaschine, daß sie eine (Zeile) 10, wohin sie gehen (GOTO) sollte, nicht gefunden hat. Kein Wunder, weil wir noch keine Zeile 10 eingegeben haben.

Sie können jeden Befehl direkt eingeben. Nach RETURN wird er sofort verarbeitet. Sie können auch mehrere Befehle durch einen : getrennt eingeben und RETURN. Der Rechner führt die Anweisung/en aus und meldet sich READY. Auf diese Weise können Sie Ihren Heim-Computer wie einen ganz normalen Taschenrechner verwenden. Welche Rechenkunststücke die Maschine drauf hat, verrät Ihnen das Anleitungsbuch.

Ein Programm besteht jedoch aus einer Reihe von Anweisungen, und diese Reihe wird (in BASIC) durchnumeriert. Eine BASIC-Zeile beginnt deshalb immer mit einer Zeilennummer, dann folgt oder folgen die Anweisung/en, und beendet wird die Zeile immer mit der RETURN-Taste (EOL), damit der Rechner weiß, das war die Zeile. Er springt dann nämlich gleich an den Anfang der nächsten Bildschirmzeile und wartet auf weitere Eingaben.

Beim ATARI sind (ganzzahlige) Zeilennummern von 0 bis 32767 zulässig. Aus schmerzhafter Erfahrung machen Profis bei den Zeilennummern immer Zehnerschritte. Dann kann man später, wenn es nötig werden sollte, noch Zeilen zwischenschieben. Sie können aber die Zeilennummern ganz frei wählen. Und gleichgültig in welcher Reihenfolge Sie die BASIC-Zeilen eingeben, der Rechner ordnet sie und arbeitet sie der Reihe nach ab, beginnend bei der niedrigsten Zeilennummer.

In einer BASIC-Zeile können mehrere Kommandos gegeben werden, man trennt sie einfach mit einem Doppelpunkt (:). Fast drei Bildschirmzeilen (106 Zeichen) kann eine BASIC-Zeile lang sein. Aber darauf brauchen Sie nicht zu achten. Wenn eine BASIC-Zeile voll ist, piepst der Konsolen-Laut-

sprecher. Wenn Sie BASIC-Befehle (in erlaubter Weise) abkürzen oder die Zwischenräume anders schreiben, als es der Rechner gerne hat, beeinflußt das die Kapazität der Zeile nicht. Diese zulässigen Abweichungen werden von der Maschine intern korrigiert.

Alte Programmierhasen schreiben die Zeilen gerne so voll wie nur eben möglich, weil das Programm schneller läuft, je weniger Zeilennummern es umfaßt. Dabei muß man aber gut aufpassen. Es gibt nämlich Befehle, die sofort zu einer anderen BASIC-Zeile hinführen, der Rest der hinter einem solchen Befehl in der Zeile steht, wird dann natürlich nicht bearbeitet.

Wahre Stolpersteine gerade für den Anfänger sind die Satzzeichen (,.;:), die so unscheinbar und ähnlich aussehen, leicht zu verwechseln und doch so wichtig sind, daß an ihnen jedes Programm zerbricht. Ein Computer versteht (wie alle dummen Menschen) absolut keinen Spaß. Ein , statt einem ; und er tritt in den ERROR- Streik oder tut etwas anderes, als man eigentlich wollte. Achten Sie also besonders auf die Satzzeichen.

Ähnlich steht es mit der 0 (Null) und O (Oh!). Für einen Elektronenrechner sind das zwei paar Schuhe. Deswegen wird die Null zur Verdeutlichung, daß sie aber auch gar nichts ist, obendrein noch durchgestrichen Ø. Durch die Auflösung in kleine Punkte ist das aber bei der Schrift, die ATARI auf dem Bildschirm hinterläßt, kaum zu sehen. Und die Drucker halten sich auch nicht alle an diese Abmachung. So auch der, dem die Programm-Listings in diesem Buch zu verdanken sind. Achten Sie also auch besonders darauf, 0 und O nicht zu verwechseln.

Probieren Sie:

```
320 A=12
111 TV=M-U
7 E=F
214 G=B*A
```

LIST (und RETURN!)

1.2 Variablen-Varianten

Variablen sind Kombinationen aus Buchstaben und Ziffern, denen beliebig Werte zugewiesen werden können (A = 17 oder ZZ3 = 7,4) und die weiter verarbeitet werden können: A + ZZ3 (= 24,4).

Das erste Zeichen einer Variablen muß immer ein Buchstabe sein, danach können beliebige alpha-numerische Zeichen (aber nur Großbuchstaben) folgen. Eine Variable könnte z. B. ATARI800XLBUCH heißen. Wer Spaß daran hat, kann eine Variable so lang machen, wie eine ganze BASIC-Zeile (106 Anschläge). Arbeiten kann man damit freilich nicht.

Der ATARI kann übrigens alle diese Variablen auch unterscheiden. Andere Geräte erlauben zwar die Eingabe beliebiger Variablen, unterscheiden aber nur die beiden ersten Buchstaben oder so.

Vorsicht ist auch geboten bei Variablennamen in denen BASIC-Wörter vorkommen. Es ist beim ATARI zwar möglich, eine Buchstabenkombination als Variable zu verwenden, die einem BASIC-Wort entspricht (s.u. LET-Anweisung), es ist aber trotzdem nicht zu empfehlen.

Wenn Sie schreiben: ENDE7B = 113 führt das jedenfalls zu nichts als einem ERROR- ENDE7B (Das E wird negativ hervorgehoben, um den Fehler zu kennzeichnen).

Insgesamt dürfen in einem Programm nicht mehr als 120 verschiedene Variablen vorkommen. Für mehr ist im Speicher kein Platz vorgesehen. Hinzu kommt noch, daß sich der Rechner alle Variablen merkt, auch solche, die beim Schreiben des Programms eingeführt, dann aber wieder gelöscht wurden. Auch durch einen Tippfehler kann eine "Variable" im Speicher notiert werden. Falls Sie irgendwann einmal ein so langes Programm kreieren, daß sie mit der Variablenmenge Probleme bekommen, können Sie das (halbfertige) Programm (mit LIST) auf Cassette oder Diskette abspeichern, den Computer aus- und wieder einschalten (die Variablentabelle wird gelöscht) und (mit ENTER) das Programm neu laden. So werden Sie alle fälschlich notierten Variablen los und haben wieder etwas Platz.

Apropos Speicher: Es ist vernünftig, die Variablen so kurz

wie möglich zu halten, weil damit Speicher (die Hauptsorge aller Programmierer) gespart wird.

Der ATARI gibt sich nur mit zwei Tyen ab (Variablentypen natürlich). Numerische Variablen enthalten wie der Name schon sagt Zahlenwerte; und String-Variablen. String heißt Kette. Gemeint ist eine Zeichen- (Character) Kette, also eine Reihe von Buchstaben (klein und groß) Ziffern und (Pseudo-grafik-) Zeichen, die über die Tastatur (CONTROL-Taste) eingegeben werden können. String-Variablen werden mit einem $ als letztem Zeichen markiert, also A$ oder BUCH4$. Der String selbst wird für den blinden Rechner mit "" markiert:

BUCH$ = "Spiele Programmieren"

Ganzzahlvariablen (A%), Variablen mit sechsstelliger Genauigkeit (A!) und solche mit 16-stelliger Genauigkeit (A#) gibt es bei manchen Rechnern, die zum Vergnügen von Technikern konstruiert wurden, ein Kreativ-Computer kommt ohne sie aus.

Ganzzahlvariablen (INTeger-Variablen), also Zahlen, bei denen die Stellen hinter dem Komma (das ist beim Computer auf gut amerikanisch ein Punkt, klar!) unterdrückt werden, liefert der ATARI auf das Kommando INT:

INT (4.35) = 4

1.2.1 LET it be

Der wichtigste Befehl, der (fast) nie verwendet wird. Zumindest beim ATARI. Wieso?

Eine Variable hat nur einen Wert, wenn ihr ein Wert zugewiesen wurde. Numerische Variablen sind bis dahin grundsätzlich gleich 0. Eine Variable ist wie ein Schubkasten, in dem etwas abgelegt wird, bis man es wieder braucht. Und natürlich können sich die Inhalte der diversen Schubkästen im Lauf des Programms verändern.

Wir müssen unserem elektronischen Freund also klarmachen, daß wir in der Schublade M den Wert 132 ablegen wollen: laß M 132 sein und das heißt auf Englisch: let M be 132. Und auf BASIC tippen wir: LET M = 132.

Wer so dumm war, im Mathe-Unterricht aufzupassen, der muß jetzt erst recht auf der Hut sein, denn dieser Befehl ist keine Gleichung. Um das zu verdeutlichen, haben die Väter von BASIC den LET-Befehl in die Welt gesetzt. Weil aber die Zuweisung eines Wertes an eine Variable so verdammt oft vorkommt und die Menschen nun einmal faul sind, ist man dazu übergegangen, das LET einfach fortzulassen. Dem Computer ist das ziemlich egal. Ob mit oder ohne LET, er ordnet M den Wert 132 zu, basta.

Wenn Sie aber aus irgendeinem unerklärlichen Grund unbedingt eine Variable verwenden wollen, es gibt ja die merkwürdigsten Leute, die einem BASIC-Befehlswort gleicht oder damit beginnt (LETTER7), denn dürfen Sie auf LET nicht verzichten:

```
10 LET LET=20
20 PRINT LET*LET
```

Wenn Sie dieses Mini-Programm eintippen und dem Rechner Beine machen, indem Sie RUN (und RETURN-Taste) eingeben, dann ordnet er brav der Variablen LET den Wert 20 zu, multipliziert LET mit LET und druckt (PRINT) das Ergebnis (tatsächlich 400) aus. READY. (Falls Sie alles richtig gemacht haben). Weil die (LET-) Zuweisung keine Gleichung ist, kann man den nützlichen Schubladen-Variablen aber auch solche Anweisungen zumuten (umblättern!):

```
Z=Z+1
```

Da fällt nun nicht nur dem Mathe-Primus auf, daß das keine Gleichung sein kann. Das Elektronengehirn erhält hier den Auftrag, die Schublade Z aufzumachen und den Inhalt von Z hineinzutun (der ja schon drin ist) und dann noch 1 dazuzutun. Schublade Z enthält dann den Wert Z+1.

Läßt man den blechernen Schlaumeier diesen Vorgang wiederholen, bringt er des doch tatsächlich fertig zu zählen!:

```
10 Z=Z+1
20 ? Z
30 GOTO 10
```

(nun mal keine Aufregung wegen dem ?, das ist nur die Abkürzung von PRINT) In Zeile 10 wird gezählt, in Zeile 20 gePRINTet, natürlich auf den Bildschirm, und Zeile 30 führt nach 10 zurück (GOTO). Wenn Sie dieses Programm RUNnen, zählt der Computer, bis er kaputt geht oder der Programmablauf unterbrochen wird (BREAK-Taste, RESET-Taste, Ausschalten). Ja so ein Computer ist doch noch folgsamer und treuer als ein Hündchen.

Der Rechner meckert auch nicht, wenn er sich den Wert, den er in einer Variablen ablegen soll, selbst ausrechnen muß:

```
10 LOLY=NUCK+ZIP/2
```

Was tut Kollege Computer bei dieser Anweisung? Er öffnet Schublade ZIP, holt den Inhalt raus und teilt ihn durch 2. Das macht er zuerst, weil ihm seine Erbauer eine Rangordnung der Rechenoperationen (Operatoren) in die Speicher gebrannt haben. Is' ja aus der Mathematik bestens bekannt, Punktrechnung vor Strichrechnung und so (nur daß das mit den Punkten bei der Punktrechnung beim Computer * / nicht so ganz stimmt).

Als nächstes öffnet er dann die Schublade NUCK und tut dann NUCK und ZIP/2 in die Variable LOLY. (Das Ergebnis ist natürlich 0, wenn wir nicht vorher NUCK und ZIP Werte zugewiesen haben.)

1.2.2 Puppet on a STRING

Nun besteht unser Leben ja nicht nur aus numerischen Werten (obwohl das die Arbeit am Computer erheblich erleichtern würde), sondern viel mehr noch aus Sprache, Text. Der Rechner kann damit gar nichts anfangen. Für ihn sind das nur irgendwelche Zeichen, Ketten von Zeichen (Englisch: String), die er mit " von seinen geliebten Zahlenwerten abgrenzt.

Das Wort "Computer" ist für den Computer also nichts als ein STRING aus Zeichen (Characters): C, o, m, p, u, t, e und r. Solche STRINGs können auch in Variablen-Schubladen verstaut werden. Um sie entsprechend kenntlich zu machen, wird dem Variablen-Namen ein $ angehängt:

```
10 DUMM$="ComPuTer"
```

Was immer zwischen den " " an Zeichen geschrieben wird, tut der Rechner in die Variable. Ihm ist völlig wurst, was das für Zeichen sind, kleine Buchstaben und große, Ziffern und Grafikzeichen, Leerzeichen oder gar Satzzeichen. Einzige Ausnahme ist das " selbst, denn das behandelt BASIC als Kennzeichen für das String-Ende. Wenn Sie einen $ löschen wollen, dann schreiben Sie:

```
20 DUMM$=""
```

Nun aber nicht so hastig!

Für Zeichenketten muß im ATARI-Speicher zuvor Platz reserviert werden und zwar für jedes Zeichen ein Plätzchen. Wenn wir für DUMM$ nur fünf Plätze reservieren, dann können wir "ComPuTer" nicht darin ablegen, weil er acht Zeichen lang ist

Das Freihalten von Speicher wird DIMensionieren genannt und der BASIC-Befehl lautet, na? Richtig! DIM. Also:

```
5 DIM DUMM$(8)
```

Ein geDIMter $ ist erst einmal leer. In Zeile 10 weisen wir ihm allerdings eine Zeichenkette zu. Deswegen können wir unseren Freund nun bitten, auf den Bildschirm zu schreiben, was er in der Schublade DUMM$ bereithält:

```
20 ? DUMM$
```

Nun DIMensionieren Sie in Zeile 5 DUMM$ auf 7 - ob das wohl einen ERROR- gibt?

Wie jede andere BASIC-Zeile kann auch die Zuordnung eines Strings nur knapp drei Bildschirmzeilen lang sein. Da in einer solchen Zeile eine Zeilennummer und die $-Variable stehen muß, kann eine String-Zuweisung (entgegen anderslautenden Gerüchten) nur maximal 99 Zeichen enthalten. Allerdings können mehrere Strings aneinandergehängt werden (wenn entsprechend geDIMt wurde). Läßt man einen solchen String auf den Bildschirm drucken, können mehr als drei Zeilen erscheinen.

Neben dem Verketten kann man Strings auch teilen, Stücke herausnehmen, die Länge abzählen und zwei Strings miteinander vergleichen:

```
A$=B$
A$<B$
A$>B$
```

Ob ein String "größer" oder "kleiner" ist, wird von der Reihenfolge des Alphabets bestimmt und zwar erst die Großbuchstaben von A bis Z, dann die Kleinbuchstaben von a bis z (ä, ö, ü und ß gehören natürlich nicht in diese Reihe, wo kämen wir denn da hin!). Noch "größer" als die Großbuchstaben sind die Ziffern von 0 bis 9.

Diese Rangordnung kommt dadurch zustande, daß der Computer jedem Zeichen einen Zahlenwert von 0 bis 255 zuordnet. Diese Zuordnung ist normiert und heißt ASCII-Standard (American Standard Code for Information Interchange).

Bei ATARI ist diese Tabelle (wie könnte es anders sein) etwas modifiziert und heißt sinnigerweise ATASCII-Code. Eine Tabelle mit dem ATARI-Zeichensatz und den zugehörigen Codes finden Sie im Anhang.

1.2.3 Andere DIMensionen

Im Gegensatz zu $-Variablen brauchen numerische Variablen nicht DIMensioniert zu werden. Oft ist es jedoch nützlich, über eine geordnete Batterie von Variablen zu verfügen. So eine Reihe von Schubkästen heißt ARRAY (= Schlachtordnung) und so ein ARRAY muß DIMensioniert werden:

```
10 DIM AR(4)
```

reserviert im Speicher Platz für fünf Variablen (fünf? Ja, fünf. Die Null wird hier mitgezählt, bei der Dimensionierung von String-Variablen nicht. Man gewöhnt sich dran - mit der Zeit) nämlich

AR(0) - AR(1) - AR(2) - AR(3) - AR(4)

Praktisch ist das deshalb, weil die Schubladen dieses Array von einer anderen Variablen gezogen werden können. Wie das?

```
20 V=V+1
30 AR(V-1)=V*V
40 GOTO 20
```

Was ist da los? In Zeile 20 taucht die Variable V auf. Sie ist erst einmal 0, ihr wird aber gleich 1 hinzugezählt, so daß in Schubalde V eine 1 drinliegt.

In Zeile 30 wird jetzt eine Schublade aus der Reihe AR (die wir in Zeile 10 geDIMt haben) ausgesucht und zwar die Nummer V-1, das ist zur Zeit (1-1=) 0, also ist die Variable AR(0) gemeint.

In Zeile 40 bitten wir den Rechner höflich, doch zur Zeile 20 zurückzukehren und weils so schön war, alles noch einmal zu machen. Nur - in der Schublade V liegt jetzt eine 1 drin und wenn in Zeile 20 eine 1 hinzugezählt wird, dann ergibt das? Und in Zeile 30 wird bei diesem Durchlauf die Schublade AR(1) mit V*V (2*2) gefüllt.

Wenn Sie das Programm laufen lassen (mit RUN und RETURN-Taste, sie wissen ja), dann gibt es allerdings einen ERROR-9 in Zeile 30. Halt! Überlegen Sie einmal selbst, was da nicht in Ordnung ist.

Wenn Sie sehen wollen, was in den Schubladen AR(0) bis

AR(4) drin ist, schieben Sie diese Zeile ein:

```
35 ? AR(V-1)
```

An dem ERROR- 9 ändert das nichts, aber Sie können jetzt wenigstens die Quadratzahlen von 1 bis 25 auf dem Monitor genießen.

Na, für alle Fälle will ich die Katze aus dem Sack lassen: Wenn das Programm nach dem fünften Durchlauf in Zeile 40 nach 20 zurückspringt, nimmt V den Wert 6 an. Na und? - In 20 wird dann die Variable AR(V-1), und das ist AR(5), aufgerufen und die gibt es nicht, weil wir in Zeile 10 AR nur auf 4 DIMensioniert haben. AR(4). Ändern! Sie die Zeile 10 in:

```
10 DIM AR(5460)
```

(mehr macht der gute ATARI nicht mit) und Sie können die Quadratzahlen von 1 bis (solange das Programm läuft können Sie ja gemütlich einen Kaffee trinken, das muß ja auch mal sein) 29.822.521 über den Bildschirm flimmern sehen. Für die 5461 Berechnungen braucht der fleißige Blechkamerad schon ein paar Minuten (BASIC-Programme sind eben doch sehr langsam!).

Die Zeile, in der DIMensioniert wird, steht gewöhnlich am Anfang des Programms, denn es muß unbedingt vermieden werden, daß diese Zeile zweimal bearbeitet wird. Ändern Sie Zeile 40:

```
40 GOTO 10
```

und nach RUN, zack: ERROR- 9 (DIM-Fehler) - eine DIMensionierte Variable darf nicht noch einmal geDIMt werden.

DIMensionierte Variablen enthalten nicht wie die einfachen numerischen automatisch eine 0. Für die zu DIMensionierende Variable wird (irgendwo) im Speicher Platz gemacht und es kann sein, daß da noch alte Werte herumgeistern, die dann in der Variablen wieder auftauchen. Dadurch können Fehler auftreten. Man sollte also jede DIMensionierte Variable (jede einzeln) auf 0 setzen. Wie man das mit wenig Anstrengung erreicht wird in Abschnitt 1.3.7 verraten.

Wenn mehrere Variablen DIMensioniert werden sollen, kann das in einer BASIC-Zeile geschehen:

```
10 DIM AA(5),BB(7),AC2(120),S$(1)
```

Natürlich können nur so viele Schubladen durch DIM reserviert werden, wie im Computer (Speicher-) Platz ist. Wenn

man ihn hier überfordert, kontert er mit ERROR- 17.

Mit dem Befehl CLR (CLeaR) können alle vorher DIMensionierten Variablen gelöscht werden. Aber aufgepaßt! Gleichzeitig werden alle übrigen, nicht DIMensionierten Variablen auf 0 gesetzt.

Neben der einfachen Schubladen-Reihe steht für besondere Zwecke auch noch die Schrankwand zur Verfügung, also ein ganzes Feld von Schubladen, das in zwei Richtungen durchnumeriert ist. Zur besseren Unterscheidung spricht man von ein- und zweidimensionalen Arrays.

Die zweiDIMensionale Variante hat die Form F(U,V), wobei statt F jeder andere Variablennamen stehen kann (nur kein $) und an Stelle von U und V können Werte, Variablen oder zu berechnende Größen stehen. Das Feld F(U,V) reicht von F(0,0) bis F(U,V), wobei in der einen Richtung von F(0,0) bis F(U,0), in der anderen von F(0,0) bis F(0,V) gezählt wird:

F(0,0)	F(1,0)	F(2,0)	F(3,0)	F(4,0)	...	F(U,0)
F(0,1)	F(1,1)	F(2,1)	F(3,1)	F(4,1)	...	F(U,1)
F(0,2)	F(1,2)	F(2,2)	F(3,2)	F(4,2)	...	F(U,2)
...	...	...	...	...	...	...
F(0,V)	F(1,V)	F(2,V)	F(3,V)	F(4,V)	...	F(U,V)

Auch bei den zweidimensionalen Arrays kann man nicht beliebig aus dem Vollen schöpfen. Selbst die künstliche Intelligenz ist bisweilen beschränkt. Das Produkt aus den beiden DIMensionierungs-Werten darf nicht größer sein als die 5461, die uns schon beim eindimensionalen Array in die Quere gekommen sind. (Vergessen Sie nicht die 0 mitzuzählen!)

DIS(12,15) müssen Sie multiplizieren: 13 mal 16.

Folgende DIMensionierungen sind z.B. möglich:

```
DIM DS1(2729,1)          (Produkt: 5460)
DIM DS2(1819,2)          (Produkt: 5460)
DIM DS3(1364,3)          (Produkt: 5460)
DIM DS4(545,9)           (Produkt: 5460)
DIM DS5(217,24)          (Produkt: 5450)
DIM DS6(108,49)          (Produkt: 5450)
DIM DS7(90,59)           (Produkt: 5460)
DIM DS8(73,72)           (Produkt: 5402)
```

Sie können auch DIM DS0(5460,0) schreiben, nur die "0" können Sie sich sparen, dann das ist ein eindimensionaler Array.

1.3 Ein paar Brocken BASIC

Eine Programmiersprache vermittelt zwischen den menschlichen Sprachgewohnheiten und dem Horizont der Maschine, der mit EIN und AUS ein nahes Ende findet; Spannung oder keine Spannung, beschrieben mit den Ziffern 1 und 0. Man spricht von höheren Programmiersprachen, je weiter sie sich von diesem Gestammel aus 0en und 1en entfernen.

BASIC ist unserer (amerikanischen) Umgangssprache sehr nahe und entsprechend weit von der binären (= zweihaften, nämlich 0/1) Form entfernt. Der erste Arbeitsgang des Rechners besteht darin, den BASIC-Befehl in Maschinen"sprache" (welch ein Wort!) umzuwandeln, also in einen Rattenschwanz aus 0en und 1en.

Diese Arbeit erledigt der BASIC-Interpreter (Übersetzer). Bei den älteren ATARI-Modellen mußte das BASIC-Modul in den Cartridge-Schacht eingesetzt werden, bei den Geräten der XL-Serie ist der Interbreter in die Innereien des Rechners integriert.

Die Übersetzung aus dem BASIC kostet Kapazität und vor allem Zeit, und wenn es auch nur Mikro-Sekunden sind, es macht sich bemerkbar. Hinzu kommt, das BASIC manchen Umweg nehmen muß, wo die Maschinensprache geradeaus gehen könnte. Deswegen kann ein Maschinenspracheprogramm um ein Vielfaches schneller laufen als das gleiche Programm in BASIC, nur - BASIC läßt sich schnell mal lernen ...

Wenn Sie also mehr aus Ihrem ATARI herausholen wollen, dann geht das nur über direkten Speicherzugriff. Darüber wird an der einen oder anderen Stelle noch zu sprechen sein. ATARI-BASIC hat zwei Befehle (POKE und PEEK), die erlauben, im Rahmen eines BASIC-Programmes einzelne Veränderungen direkt in die Windungen der Schaltkreise zu schreiben. Damit läßt sich dem Computer manches Kunststück beibringen, daß ihm BASIC nicht erklären kann.

Jetzt sollen Sie sich aber erst einmal mit den wichtigsten Befehlen vertraut machen.

1.3.1 PRINTen mit?

Der PRINT-Befehl veranlaßt den Computer zu einer Ausgabe, die normalerweise auf dem Monitor erfolgt. Es kann das Ergebnis einer Berechnung (PRINT 21*16+7) oder der Inhalt einer Variablen (PRINT A$ / PRINT V) ausgegeben werden. Soll die Ausgabe nicht auf dem Bildschirm sondern an anderer Stelle erfolgen, z.B. auf einem Drucker (sofern man einen hat und selbiger angeschlossen ist), muß der PRINT-Befehl entsprechend erweitert werden.

Wenn die Ausgabe abgeschlossen ist, springt der Cursor (Schreibmarke) an den Anfang der nächsten Zeile und die Maschine ist für neue Eingaben bereit.

Nun probieren Sie bitte folgendes:

```
10 DIM N$(14)
20 N$="Renate Knuefer"
30 PRINT N$
40 GOTO 20
RUN
```

Um dem endlosen Spiel ein Ende zu bereiten, drücken Sie die BREAK-Taste (oben rechts). Es erscheint die Meldung STOPPED AT LINE ... Tippen Sie jetzt LIST 30 (RETURN), dann bekommen Sie Zeile 30 ausgeLISTet, und setzen Sie hinter PRINT N$ ein ; - sehen Sie sich an, was Sie nach RUN erwartet.

Ersetzen Sie jetzt das ; durch ein , !

Der Beistrich (Komma) läßt den Cursor nach dem Ausdruck zum nächsten Tabulator-Stop springen (das kennen Sie doch von der Schreibmaschine, oder?) Beim Einschalten sind TAB-Stops auf die Spalten 7, 15, 23, 31 und 39 gesetzt. Bringen Sie den Cursor in eine beliebige Spalte und löschen Sie einen gesetzten TAB-Stop durch gleichzeitiges Drücken der Tasten TAB und CONTROL (CTRL) oder setzen Sie einen TABulator-Stop durch Tasten TAB und SHIFT.

Weil der PRINT-Befehl recht oft verwendet wird, kann er durch ? ersetzt werden. Ein ?- oder PRINT-Befehl ohne weitere Anweisung führt zur Ausgabe einer Leerzeile. ?: ?: ?

1.3.2 I. wie INPUT

Nehmen wir einmal an, Sie haben ein wirkliches mathematisches Problem, das auf Ihnen lastet, so eine richtige Aufgabe für einen Computer. Vielleicht brauchen Sie häufig das Quadrat von irgendwelchen Zahlen. Sowas soll es ja geben. Natürlich können Sie jedesmal die ganze Rechenaufgabe eintippen. Wenn die Zahl 9 lautet also:

```
? 9*9   (RETURN)
```

Promt kommt die Antwort:

```
81
READY
```

Ja und dann können Sie für die nächste Zahl wieder den ganzen Befehl eintippen. Aber da stimmt doch irgendetwas nicht? Für diese Stumpfsinnsarbeiten haben wir doch den Blechkameraden. Wir müßten ihm nur erklären: Hier, mein Freund, das ist die Zahl und nun sag mir dazu das Quadrat. Kein Problem?

```
100 A=9
200 ? A*A
```

Prima! und wenns nun nicht 9 sondern 12 sein soll?

Wir müßten das Programm dazu bringen, in Zeile 100 innezuhalten und darauf zu warten, bis der Herr und Meister (das sind wir natürlich) die zu quadrierende Zahl eingibt. Genau das nennt man einen INPUT (Eingabe):

```
100 INPUT A
```

Wenn Sie das Programm jetzt RUNnen lassen, erscheint auf dem Bildschirm ein forderndes ? - man wartet auf Ihren INPUT. Jetzt können Sie nach Herzenslust einen Wert eintippen, z.B. 352,8327 oder so, und es geschieht überhaupt nichts. Sie müssen Ihren INPUT nämlich durch betasten von RETURN beenden.

Sofort erscheint der gesuchte Wert: 123904. Und Sie freuen sich auch noch? Der Wert stimmt doch überhaupt nicht. Die Wurzel aus 123904 ist exakt 352. Wo sind die Stellen hinter

dem Komma - ah, ja ... wo sind wir denn hier! Stellen hinter dem Komma gibt es nicht. Sie befinden sich auf amerikanischem Territorium. Also passen Sie sich gefälligst der Landessitte an. Statt des Kommas ist ein Dezimalpunkt zu verwenden (als kleines Trostpflaster dürfen Sie auch die 0 vor dem Punkt fortlassen, also .6 statt 0.6, klar?)

Also nochmal RUN und diesmal: 352.8327. Zack: 124490.9141 - prima, was! Wenn Sie jetzt noch die Zeile

```
300 GOTO 100
```

an das Programm anhängen, dann können Sie endlos Werte eingeben und bekommen schwupp-di-wupp das Quadrat. Nur eines dürfen Sie nicht: etwas anderes als Ziffern eingeben, wenn das ? erscheint.

Natürlich kann der Rechner mit INPUT auch Strings aufnehmen. Der Befehl lautet dann entsprechend INPUT BLABLA$. Natürlich muß BLABLA$ vorher entsprechend DIMensioniert worden sein:

```
100 DIM BLABLA$(30)
110 ? :? "Bitte geben Sie Ihren Namen ein:":?
120 INPUT BLABLA$
130 ? :? "DANKE!"
140 ? :? "Ihr Name ist "
150 ? :? LEN(BLABLA$)
160 ? :? "Buchstaben lang"
170 GOTO 100
```

100: DIMensioniert die $-Variable vorsorglich auf 30 Zeichen. Sollte irgendwer einen längeren Namen haben oder aus bloßem Trotz mehr Zeichen INPUTten, dann gibts natürlich ERROR-Probleme.

110: Die vielen Fragezeichen ergeben Leerzeilen und lockern damit das Schriftbild auf. Benutzerfreundlich nennt man das.

120: Hier erscheint, wenn das Programm abläuft das INPUT-? und der Name kann eingegeben werden (RETURN nicht vergessen). Natürlich können auch Ziffern, Zeichen und Leerstellen eingetippt werden. Die Maschine schluckt die Zeichenkette wie sie ist, ohne darüber nachzudenken. Denken Sie mal!

130: Leerzeile und eine kleine Freundlichkeit.

140: s. Kommentar zu Zeile 130.

150: Ja, was haben wir denn da? Was soll da ausgedruckt werden: LEN(BLABLA$)?*

160: s. Kommentar zu 130.

170: Führt das Programm an den Anfang zurück, damit wir das Spielchen noch einmal genießen können.

Also RUN an den Speck! Und: ERROR- 9 AT LINE 100? In Abschnitt 1.2.3 nachlesen, am Kopf kratzen und Zeile 170 ändern in GOTO 110 oder CLR: GOTO 100.

Nur so zum Spaß sollten Sie jetzt in Zeilen 140 und 150 ein ; ans Ende setzen. Mal RUN. Sieht doch besser aus, oder? Nur den $ in 160 sollten Sie jetzt noch mit einem Leerzeichen beginnen lassen, um den fehlenden Zwischenraum zu korrigieren. (Wenn Sie nicht verstehen, wovon ich rede, dann müssen Sie die Beispiele eben doch ausprobieren)

Mit einem INPUT können auch mehrere numerische Werte und/oder Strings aufgenommen werden:

```
1000 INPUT A,B$,C$,D,E
```

Der Benutzer muß nur genau wissen, was er in welcher Reihenfolge eingeben soll, d.h. Ihr Programm sollte es ihm sagen. Wird ein $ erwartet aber eine Zahl eingegeben, dann verarbeitet der Rechner die Ziffernfolge als Zeichenkette. Wird aber ein numerischer Wert erwartet und der Benutzer drückt versehentlich (oder vorsätzlich) eine Buchstabentaste, gibts ERROR- 8 (Input in falsche Variable) und der programmierunkundige Benutzer steht ratlos vor einem abgebrochenen Programm.

* der Befehl LEN(X$) zählt die Anzahl von Zeichen in der Kette X$. Dabei wird alles mitgezählt, also auch die Leerzeichen = Zwischenräume. Was abgezählt werden soll steht in der ( ) hinter LEN, das kann eine $-Variable sein (LEN (BLABLA$)) oder ein String selbst (LEN(Zaehl STRING)) (=12)

1.3.3 O gott, Otto: GOTO

GOTO heißt auf gut Deutsch GENACH. Muß ich noch mehr erklären?

Na gut.

Auf das Kommando GOTO springt das Programm sofort (!) zu der hinter GOTO angegebenen Zeilennummer und arbeitet diese ab. Also erstens: Es muß eine solche Zeile geben, sonst springt das arme Programm ins Leere und revanchiert sich mit einem ERROR- 12 (Zeile fehlt); und zweitens werden Befehle, die womöglich in der gleichen Zeile noch hinter GOTO stehen, natürlich nicht ausgeführt, weil das Programm abspringt, bevor es dorthin kommt.

Mit GOTO lassen sich hübsche Spielchen mit dem Computer treiben, der ja nicht weiß, wie ihm mitgespielt wird:

```
100 ? "ICH LIEBE DICH. ";
200 GOTO 100
RUN
```

(Ach tut das gut!)

In Zeile 100 bitte die Leerzeichen hinter dem . nicht einsparen und auch das ; hat seinen Wert, dort wo es steht.

Statt eines Zahlenwertes kann hinter GOTO auch eine numerische Variable (A) oder ein zu berechnender Wert (5+19 / B*3) stehen. Etwaige Dezimalstellen eines errechneten GOTO-Sprungzieles werden unterdrückt. Hauptsache ist nur, daß die jeweils errechnete und angesprungene Zeilennummer vorhanden ist. (Muß ich extra erwähnen, daß zu jeder beliebigen Zeile, vorwärts wie rückwärts, gesprungen werden kann?)

GOTO kann auch als Direkteingabe, also ohne Zeilennummer, verwendet werden. GOTO 250 (RETURN) springt in Zeile 250 und beginnt damit, diese und dann die folgenden abzuarbeiten. Voraussetzung ist natürlich, daß ein Programm im Computer gespeichert ist, das eine Zeile 250 aufweist und daß es möglich ist, die Abarbeitung des Programms in Zeile 250 zu beginnen, ohne daß ERROR- dazwischenfunkt.

Noch mehr GOTO-Spielereien.

Neben dem großen GOTO-Sprung, beherrscht der ATARI auch das Kunststückchen des bedingten Sprunges. Die Zauberformel dafür lautet: ON (Bedingung) GOTO (Zeilennummer/n):

```
300 ON A=B GOTO 400
```

Ist A nicht gleich B wird der Sprung nicht ausgeführt und das Programm macht mit der folgenden Zeile weiter.

Der ON-GOTO-Befehl kann aber auch so aussehen:

```
310 ON A GOTO 200,220,240,100,310,700
```

(hinter GOTO können so viele Zeilennummern folgen, wie die BASIC-Zeile verkraftet)

Die numerische Variable (A) bestimmt, welche Zeilennummer für den Sprung ausgewählt wird. Ist A=0, wird der GOTO-Befehl übergangen. Hat A einen Wert zwischen 1 und 255 (Dezimalstellen werden ggf. unterdrückt), werden die aufgereihten Sprungziele hinter GOTO abgezählt. A=3 führt zu einem Sprung in die Zeile deren Nummer als dritte hinter GOTO steht, in der obigen Beispielzeile ist das 240. Hat A einen Wert, der größer ist, als die Anzahl der aufgereihten Sprungziele, wird eben nicht gesprungen.

Ist A kleiner 0 oder größer 255 setzt es eine ERROR-feige.

Die ON-Bedingung kann auch auch ein zu berechnender Wert sein:

```
100 ON A=4 GOTO 700
200 ON A*A GOTO 800
300 ON A/Z GOTO 900
400 ON A+G<B GOTO 150
500 ON A*6>B+5 GOTO 160
600 ON A(I-7) GOTO 100,200,300,400
700 ON R(U,6) GOTO 110,120,130,700
```

ON kapiert GOTO nächstes Kapitel

1.3.4 GOSUBkultur

Ein weiterer Sprungbefehl, der Folgen hat. GOSUB. Auch hier springt das Programm in die angegebene Zeilennummer. Den angesprungenen Teil nennt man Unterprogramm, ein in sich geschlossener Programmteil, der vom Hauptprogramm nur über den GOSUB-Sprung erreicht wird. Das Unterprogramm endet mit dem Befehl RETURN (das ist etwas anderes als die RETURN-Taste, klar doch) verlassen. Das Programm fährt dann in der Zeile fort, die direkt der Zeile folgt, die mit GOSUB verlassen wurde:

```
10 A=27
20 GOSUB 2000
30 END
2000 ? A
2010 RETURN
RUN
```

10: ordnet der numerischen Variablen A einen Wert zu.

20: Sprung ins Unterprogramm ab Zeile 2000.

30: Programmende. Muß extra programmiert werden, um das Hauptprogramm vom Unterprogramm zu trennen. Der Rechner arbeitet ein Programmzeile nach der nächsten ab, er würde also nach Zeile 20 in Zeile 2000 weitermachen, wenn ihm hier nicht Einhalt geboten würde.

2000: Hier beginnt das Unterprogramm. Der Wert von A wird ausgedruckt.

2010: verordnet den Rücksprung in die Zeile hinter GOSUB und das ist in diesem Beispiel 30 ENDe.

Natürlich kann man aus einem Unterprogramm in ein Unter-Unterprogramm springen (und so weiter, wenn es Spaß macht). Man läuft nur Gefahr, daß man die Programmstruktur nicht mehr durchschaut und sich in Fehlern verzettelt.

Noch ein Wort zu RETURN.

Ein Unterprogramm sollte immer mit RETURN verlassen werden, weil sich der Rechner irgendwo in den unergründlichen Tiefen seiner Schaltkreise die Zeilennummer gemerkt hat, bei

der GOSUB angeordnet wurde, um irgendwann dorthin zurückzuRETURnen. Wenn aber kein RETURN kommt, um die gespeicherte RETURN-Zeile zu erlösen, dann irrt dieser Wert durch den Speicher und richtet vor lauter Verärgerung womöglich Unheil an.

Wenn es bei irgendeinem wahnsinnigen Programm trotzdem einmal nötig sein sollte, mitten aus dem Unterprogramm auf Nimmerwiedersehen herauszuspringen, dann sollte der Chip mit POP beruhigt werden. Ein solches Unterprogramm könnte so aussehen (aber auch anders):

```
1000 INPUT WERT
1010 IF WERT>20 THEN ? "zu gross":POP :GOTO 20
1020 ? "O.K."
1030 RETURN
```

1000: erwartet eine numerische Eingabe, die der Variablen WERT zugeordnet wird.

1010: druckt den String "zu groß" (aber nur, wenn WERT größer als 20 ist, und da sind wir doch schon mitten im nächsten Kapitel), entschuldigt sich dann mit POP für einen überraschenden Abgang und springt nach Zeile 20 (es ist vorteilhaft, wenn es im Hauptprogramm eine Zeile 20 gibt)

1020: Wenn WERT nicht größer als 20 war gibt es weder ? noch POP noch GOTO und das Programm läuft hier weiter: der String "O.K." wird auf den Monitor ge?tet und mit

1030: gehts RETURN ins Hauptprogramm (und zwar zu der Zeile, die der Zeile folgt, in der der Sprung aus dem Hauptprogramm in das Unterprogramm mit GOSUB usw.)

Übrigens: Was GOTO recht, ist GOSUB billig. Sprungziele können numerisch, als Variable oder zu berechnende Werte verabreicht werden. Auch mit ON steht GOSUB auf Du und Du.

1.3.5 Wenn IF dann THEN

Der Computer trifft eine Entscheidung! Hier offenbart sich das intimste Geheimnis der künstlichen Intelligenz (was für ein dummes Wort). Allerdings - wenn die Maschine eine Entscheidung treffen soll, dann muß der Programmierer (mit seiner natürlichen Intelligenz) jede Möglichkeit vorhersehen und für jede Bedingung (IF) eine Folge (THEN) festsetzen.

Wenn im Programmablauf ein IF auftritt, überprüft der Rechner die dahinter stehende Bedingung. Das IF kann sich auf einen String beziehen (IF N$="14"), auf eine numerische Variable (IF N=14) oder auf einen zu berechnenden Wert (IF A+14 < B. (Und auch auf boolesche Ausdrücke. Fragen Sie hierzu Ihren Computer-Fachhändler)

IF die Bedingung erfüllt ist THEN wird die Anweisung ausgeführt, sonst wird sie ignoriert. Die Bedingung kann ein Sprungziel sein (THEN 200, GOTO kann hier fortfallen, muß aber nicht), auch ein berechnetes (THEN GOTO V+66) aber auch jeder andere Befehl ist denkbar (THEN ?"Prima" / THEN A=A+23 / THEN END / THEN GOSUB 20000 usw.)

Statt einer Bedingung kann auf THEN aber auch ein weiteres IF-THEN folgen. So können IF-THEN-Bedingungen verkettet werden, bis die BASIC-Zeile aus dem Bildschirm platzt:

```
300 IF A=B THEN IF C<D THEN IF E<>0 THEN
```
und so weiter

Nur an eines sollten Sie immer denken: IF Bedingung erfüllt - THEN sofortige Ausführung und IF Bedingung nicht erfüllt - THEN sofortiger Übergang zur nächsten BASIC-Zeile.

Wenn die Bedingung also ein Sprungbefehl (GOTO oder GOSUB) ist, können in der BASIC-Zeile dahinter keine weiteren Kommandos gegeben werden:

```
400 IF A=0 THEN 100:? "Abseits"
```

Ob die Bedingung zutrifft oder nicht, das Programm gelangt nie zu dem PRINT-Befehl.

```
500 IF A=0 THEN A=B:? "Pech"
```

Aufgepaßt! Wenn die Bedingung nicht zutrifft, wird sofort die nächste Zeile bearbeitet und "PECH" hat Pech gehabt.

1.3.6 FOR the NEXT STEP: Schleifen

Sie erinnern sich? In Abschnitt 1.2.3 war ich zu faul, jede einzelne indizierte Variable durch eine gesonderte Eingabe auf 0 zu setzen (natürlich nur, weil ich wußte, daß es auch anders geht). Für solche Routinearbeiten ist die FOR-NEXT-Schleife erfunden worden.

Mit Schleife ist gemeint, daß ein bestimmter Programmteil mehrmals durchlaufen wird. Die Schleife beginnt mit FOR. Hinter FOR muß ein Schleifenzähler definiert werden, der die Durchläufe zählt:

```
500 FOR Z=0 TO 20
```

Z ist der Schleifenzähler (der kann natürlich auch anders heißen). Jede beliebige numerische Variable kann verwendet werden, und der Anfangs- und der Endwert für den Schleifenzähler können als Dezimalwert, als Variable oder als zu berechnender Wert eingegeben werden:

```
510 FOR SCH=A TO B*C
```

Der Schleifenzähler in Zeile 500 zählt von 0 bis 20, die Schleife, in der beliebig viele BASIC-Zeilen und -Befehle stehen können, wird also 21mal durchlaufen. Das Ende der Schleife wird durch NEXT und der Schleifenvariablen markiert:

```
550 NEXT Z
```

Solange die Schleifenvariable nicht den festgelegten Höchstwert erreicht hat, erfolgt bei NEXT der Rücksprung in die zugehörige FOR-Zeile.

```
700 FOR J=0 TO 5
710 A(J)=0
720 NEXT J
```

Was geschieht? In Zeile 700 wird J auf 0 gesetzt, in Zeile 710 wird A(J), also A(0) mit 0 gefüllt und in Zeile 720 wird nach 700 zurückgesprungen. J bekommt jetzt den Wert 1, in 710 wird A(1) auf 0 gesetzt usw. Wenn J den Wert 5 erreicht hat, wird in 720 kein Rücksprung ausgelöst, das Programm arbeitet die BASIC-Anweisung mit der nächst höheren Zeilennummer ab.

Wird nichts anderes programmiert, zählt die Schleifenvariable bei jedem Durchlauf 1 höher. Diese Schrittweite läßt sich jedoch auch bestimmen:

```
800 FOR Q=12 TO 24 STEP 2
```

zählt nur jeden zweiten Wert. STEPs können Dezimalstellen haben (STEP 1.33), Variablen sein (STEP F) oder zu berechnende Werte (STEP W/3). Das schlaue Kerlchen kann sogar rückwärts zählen:

```
850 FOR D=100 TO 20 STEP -10
```

zählt 100, 90, 80, ... bis 20.

Der Clou ist aber, daß in einer FOR-NEXT-Schleife weitere FOR-NEXT-Schleifen liegen können:

```
10 DIM F(10,20)
20 FOR X=0 TO 10
30 FOR Y=0 TO 20
40 F(X,Y)=0
50 NEXT Y
60 FOR W=0 TO 500:NEXT W
70 NEXT X
80 ? "Fertig"
```

10: DIMensioniert die Variable F auf (10,20). Um nun alle Einzelvariablen des Arrays auf 0 zu setzen, bedienen wir uns einer verschachtelten FOR-NEXT-Schleife.

20: eröffnet die Schleife der Zählvariablen X.

30: eröffnet die Schleife der Zählvariablen Y

40: beim ersten Durchlauf haben X und Y den Wert 0, F(0,0) wird also auch 0 gesetzt.

50: hier signalisiert NEXT Y den Rücksprung nach Zeile 30, wo der nächste Wert für Y genommen wird. X ist noch immer 0. In diesem Durchlauf wird also F(0,1) auf 0 gesetzt. Erst beim 21. Durchlauf, F(0,20)=0 gibt NEXT Y den Weg frei.

60: auch so etwas ist möglich, eine leere Schleife. W wird von 0 bis 500 gezählt, dann erst gibt NEXT W grünes Licht.

70: hier ist das Ende der X-Schleife und die ist noch lange nicht abgearbeitet. Also zurück nach 20, X bekommt den Wert 1 und dann kommt wieder die Y-Schleife an die Reihe. Bis die X-Schleife vollkommen abgearbeitet ist (11 Durchläufe) wird die Y-Schleife 11mal abgearbeitet. Auf diese Weise werden alle Einzelvariablen F(X,Y) auf 0 gesetzt.

(Irgendwelche Fragen zu Zeile 80?)

1.4 Ab hier geht's bunt zu

So, das wäre geschafft.

Die bislang behandelten BASIC-Befehle sind bei allen Heim-Computern recht ähnlich. Unterschiede gibt es hauptsächlich bei den Abkürzungen für die Befehle oder beim Zeilenformat. Einfachere Geräte lassen z.B. nur einen Befehl pro BASIC-Zeile zu.

Mit dieser handvoll Wörtern können Sie schon eine ganze Menge ausrichten, wie wir noch sehen werden. Alle Befehle, der ATARI kennt knapp 80, die noch hinzukommen, werden entweder sehr selten gebraucht, beziehen sich auf Zusatzgeräte ("Peripherie"!), wie Diskettenstation, Drucker und so oder sind wirklich erste Sahne und eigentlich schon kein BASIC mehr (PEEK, POKE, USR usw.)

In diesem Kapitel gibt es ein paar Befehle zu bestaunen, die den ATARI von allen Geräten dieser Preisklasse abheben. Das sind Befehle für Grafik und Farbe. Einem Unternehmen entsprungen, das mit Telespielen weltweit großen Erfolg hat, wurde dem ATARI-Computer einiges in die Schaltpläne geschrieben, was das Auge erfreut.

Wer Kreatives im Programmier-Schilde führt, dem wird es mit diesem Hausfreund besonders leicht gemacht.

1.4.1 GRAPHICS-GR.affiti

Wenn Sie Ihren ATARI-Computer einschalten (haben Sie eigentlich einen Computer, oder lesen Sie nur so?). Also wenn Sie ihn einschalten können, dann ist die Grafikbetriebsart 0 eingestellt, mit dem Text auf den Bildschirm gebracht wird, 40 Spalten, 24 Zeilen und in langweilig schönem Hellblau.

Wenn es um die Darstellung von Grafik und Text geht, dann ist der ATARI ein rechter Wunderknabe. Mit dem Befehl GRAPHICS, sinnvollerweise GR. abzukürzen, lassen sich zwölf verschiedene Darstellungsarten (GR.0 bis GR.11) abrufen. Für diese Bildfülle ist ein spezieller Chip verantwortlich, der sich unter dem Gehäuse des ATARI versteckt und ANTIC gerufen wird. ANTIC ist klug und eigentlich versteht er nicht nur zwölf sondern sogar sechzehn Anweisungen. Und tatsächlich lassen sich bei den ATARI-XL-Modellen auch die Befehle GR.12 bis GR.15 erteilen.

ANTIC ist immer noch der alte, aber das ATARI-BASIC ist entsprechend erweitert worden. Bei der Gelegenheit sind leider auch ein paar weitere Änderungen angebracht worden. So ist bei den älteren Geräten COLOR durch CO. abzukürzen, bei den neueren nur durch C. Da müssen Sie etwas aufpassen.

Im Verlauf eines Programms können beliebig viele und verschiedene GR.-Befehle stehen. Ein GRAPHICS-Befehl löscht jedoch als erstes den Bildschirm. Es ist also nicht möglich (in BASIC) verschiedene Grafikmodi auch einem Bildschirm zu mischen. Daß das doch geht, sieht man bei vielen Fix-und-Fertig-Computer-Spielen, aber um das zu programmieren, muß eine Display-List erstellt werden und vieles über Video-Technik und die Computer-Innereien gewußt werden, und das ist ein Thema für ein extra Buch.

Um das Mischen von Grafik und Text trotzdem ein wenig zu ermöglichen, kann in die reinen Grafikbetriebsarten am unteren Rand des Bildschirms ein Textfenster eingeblendet werden.

Bei Programmabbruch (ERROR-), Programmende, BREAK-Taste oder RESET-Taste wird auf GR.0 zurückgeschaltet.

1.4.2 MultiCOLOR

Wenn Sie einen Grafikmodus auserkoren haben, wird Ihnen eine Farbauswahl frei Bildschirm mitgeliefert. Doch damit müssen Sie sich nicht bescheiden. Ihr ATARI'Computer verfügt über einen Farbreichtum, der zum Schwärmen verleitet. Allerdings lassen sich in den meisten Betriebsarten nur zwei oder vier dieser vielen schönen Farbtöne gleichzeitig darstellen. Nur in den höheren GRAPHICS (9 bis 11) geht es bunter zu. Aber damit wird auch beengend viel Speicherplatz verbaucht, und wenn Sie nicht ein 64 kByte-Modell Ihr Eigen nennen, dann haben Sie schnell mit ERROR- 2 zu tun (Speicher voll)

Die Auswahl der Farbtöne erledigt der Setz-Farbe-Befehl (SETCOLOR), der freundlicherweise SE. abgekürzt werden darf. Der SETCOLOR-Befehl hat die Form:

SETCOLOR r,f,h

r ist eine Kennzahl von 0 bis 5 für die fünf verschiedenen Farbtöne, die mit diesem Befehl definiert werden können. f bestimmt den Farbwert (0 bis 15) und h die Helligkeitsstufe von 0 (dunkel) bis 15 (hell), dabei sind aber nur die geraden Werte, also acht Stufen zulässig. Nur GR.9 (s. 2.2.9) läßt alle 16 Schattierungen zu.

Mit dem SETCOLOR-Befehl werden die abrufbaren Farbtöne ausgewählt. Es ist aber möglich, im Verlauf des Programms die Farbtöne neu zu bestimmen. Wenn Sie keine Farbwerte definieren, bleiben die Standardwerte (s. Tabelle unten) erhalten.

Insgesamt sind also 256 Farbtöne aufrufbar. Grundsätzlich stehen aber nur 128 Farbtöne zur Verfügung (Helligkeit nur gerader Wert), wenn mehrere Farbtöne definiert werden können. In GR.9 kann nur ein Farbwert aufgerufen werden, dafür stehen dann aber sechzehn Helligkeitsstufen zur Auswahl.

Der SETCOLOR-Befehl löst im Rechner folgendes aus: Der Wert f für die Farbe wird mit 16 multipliziert und der Wert h für die Helligkeit addiert. Der so errechnete Farbwert wird in das entsprechende Farbregister geschrieben.

Ein Farbregister ist eine Speicherzelle im Gehirn der Maschine, die nur dafür reserviert ist, Farbwerte aufzunehmen und entsprechende Schaltungen auszulösen. SE.0 bis 4 bezieht sich auf die Register 708 bis 712.

Die Umrechnung des (BASIC-) Befehls SETCOLOR kostet Zeit und belegt relativ viel Speicher. Da in diesem Buch noch viel mit Grafik und Farbe gearbeitet wird, sollten Sie sich gleich daran gewöhnen, die Farbwerte direkt in die entsprechenden Register zu schreiben.

Für den direkten Speicherzugriff gibt es die Befehle POKE und PEEK. POKE r,n bringt den Wert n in die Adresse r. Mit PEEK(r) kann der Inhalt der Adresse r gelesen und z.B. mit PRINT ausgegeben werden. Schreiben Sie also statt:

SE. 0,2,8	(Orange)	POKE 708,40	(2*16+8)
SE. 1,12,10	(Hellgrün)	POKE 709,202	(12*16+10)
SE. 2,9,4	(Mittelblau)	POKE 710,148	(9*16+4)
SE. 3,4,6	(Rotviolett)	POKE 711,70	(4*16+6)
SE. 4,0,0	(Schwarz)	POKE 712,0	(0*16+0)

(dies sind die vorgegebenen Standard-Farbwerte)

Die so definierten Farbtöne werden mit dem COLOR-Befehl aufgerufen. Eine Kennzahl von 0 bis 4 hinter COLOR bezieht sich jeweils auf eine mit SE. (oder POKE) definierte Farbe. Aber um es den Computer-Freunden nicht gar zu leicht zu machen, gehören je nach GR.-Betriebsart ein anderes COLOR und SE. zusammen (ja das übt den Geist).

Nach so viel Studiereifer sollten wir uns gemeinsam zu einem schönen Farberlebnis zurücklehnen und einen kurzen Blick auf die ATARI-Palette werfen:

```
10 GRAPHICS 19
20 FOR C=0 TO 255 STEP 2
30 POKE 712,C
50 NEXT C
RUN
```

O.K., das war ein sehr kurzer Blick. So schnell kann eben selbst ein Heim-Computer sein. Schieben wir eine Pause ein:

```
40 FOR Z=0 TO 200:NEXT Z
RUN
```

Na? Wenn Sie jetzt so richtig neugierig auf Farben sind, dann schnell umblättern -

1.4.3 Farbenrausch

128 Farbtöne sind eine ganze Menge. Für die bescheidenen Ansprüche eines wenig geübten Auges kann man da schon fast von bunt sprechen. Das menschliche Auge ist zwar in der Lage, allein so knappe 50.000 Gelbtöne zu unterscheiden, aber wir wollen ja keine Ansprüche stellen.

Allerdings ist die Farbe Gelb ein besonderes ATARI-Problem. Schöne klare Gelbtöne sind in der 128-Farben-Palette nicht enthalten.

Auf der folgenden Seite finden Sie eine Tabelle mit Beschreibungen zu den Farbtönen. Unsere Sprache ist recht unbeholfen, wenn es um Farben geht. Aber die Tabelle gibt Ihnen einen ersten Anhaltspunkt. Die Farbwirkung hängt auch ganz kräftig von der Einstellung Ihres Fernsehers oder Monitors ab. Drehen Sie mal an den Knöpfchen für Kontrast und Farbtönung.

Wollen Sie sich die Farbtöne einmal auf dem Bildschirm ansehen? Dann tippen Sie doch dieses kleine Programm ein. Nach RUN können Sie durch Drücken der Taste 1 den Farbwert und durch Taste Q den Helligkeitswert der Farbe auf dem Bildschirm verändern. Im Textfenster erscheinen der Farb- und der Helligkeitswert:

```
190 GRAPHICS 3
200 OPEN #1,4,0,"K:"
210 GET #1,T
220 IF T=49 THEN CO=CO+1:IF CO>15 THEN CO=0
260 IF T=81 THEN HO=HO+2:IF HO>14 THEN HO=0
400 SO=CO*16+HO:POKE 712,SO
440 ? "FARBE ";CO;" / HELL ";HO;" / POKE ";SO
450 CLOSE #1:GOTO 200
```

Wenn Sie die Wirkung von vier verschiedenen Farben auf dem Bildschirm betrachten und die Farbtöne dabei verändern wollen, dann möchte ich Ihnen das Programm Farbenschlüssel im Kapitel 5 dieses hervorragenden Buches ans Herz legen. Wenn Sie gewählt haben, liefert Ihnen das Programm die Werte, die in die Farbregister gePOKEt werden müssen.

Schwarz	Mittel-	relativ gleichmäßige Grau-Skala bis fast					Weiß
0	2	4	6	8	10	12	14
Purpur	Rehbraun	Curry	Schmutzig-	leicht grünliche Gelbtöne			Eierschale
16	18	20	22	24	26	28	30
Rot	Indischrot	gedecktes	Orange	Orange	über Gelb bis weißlich		
32	34	36	38	40	42	44	46
Glutrot	feurig	Altrosa	Lachs	Abstufungen von Rosa			
48	50	52	54	56	58	60	62
Violett	Flieder	von Hellviolett über Lila nach weißlich Lila					
64	66	68	70	72	74	76	78
Blau	von Blauviolett über Flieder nach Blaßlila						
80	82	84	86	88	90	92	94
Cyan	gleichmäßige Skala bis fast Weiß						
96	98	100	102	104	106	108	110
Cyan	leicht graustichig	fast identisch mit der vorstehenden Reihe					
112	114	116	118	120	122	124	126
Preußisch	Skala etwas dunstiger Blautöne						
128	130	132	134	136	138	140	142
Dunkelblau	Mittelblau	von Taubenblau über Stahlblau nach Eisblau					
144	146	148	150	152	154	156	158
Mittelgrün	über Hellgrün und Giftgrün ins leicht türkisierende Weiß						
160	162	164	166	168	170	172	174
Mittelgrün	Maigrün	der vorstehenden Reihe ähnlich jedoch etwas greller					
176	178	180	182	184	186	188	190
Dunkelgrün	gleichmäßige Skala mit höherem Gelbanteil						
192	194	196	198	200	202	204	206
Moosgrün	wie vorstehend jedoch mit noch größeren Gelbanteilen						
208	210	212	214	216	218	220	222
Braun	Oliv	Grünbeige	gelblich schmutzige Grüntöne				
224	226	228	230	232	234	236	238
Karminrot	von Rehbraun über Curry und schmutzige Gelbtöne						
240	242	244	246	248	250	252	254

1.4.4 Zeichensprache

Die Kommunikation mit der Maschine erfolgt über Zeichen, Buchstaben und Ziffern. Da der Rechner nun einmal nicht mehr als EIN und AUS versteht, läßt man ihn auch alle Zeichen aus diesem reichen Schatz von Ausdrucksmöglichkeiten zusammensetzen. Da trifft es sich gut, daß Bruder Bildschirm auch nicht mehr kann: Licht an, Licht aus. Stark sind die Jungs ja nur in der Menge.

Im normalen Textmodus (GRAPHICS 0) ist der Bildschirm in 960 Schreibpositionen (40 Spalten, 24 Zeilen) aufgeteilt. Jede dieser Schreibpositionen ist acht Bildschirmzeilen hoch und acht Bildpunkte breit. Jedes Zeichen kann also nur aus 8 mal 8 Punkten zusammengesetzt werden. Für ästhetisch wenig anspruchsvolle Techniker ist das mehr als reichlich.

Alle alpha-numerischen Zeichen und noch ein paar grafische Elemente (Pseudo-, Block- oder Semigrafik geheißen), sind Pünktchen für Pünktchen festgelegt und schlummern in Zahlenwerte umgewandelt im mystischen Inneren des Elektronengehirns, von wo sie bei Bedarf, z.B. wenn Sie auf der Tastatur herumtippen, um ein BASIC-Programm zustande zu bringen, abgerufen werden. Wenn Sie Lust haben, sich diese Punktmuster der Zeichen vergrößert anzusehen, dann empfehle ich Ihnen das Programm Bit-Muster in Kapitel 5.

Die Zeichen des Zeichensatzes sind übrigens alle kleiner als das 8 mal 8 Feld der Schreibposition. Sie können das gut sehen, wenn Sie den Cursor über ein Zeichen setzen, denn der füllt die Schreibposition völlig aus.

Wer's kann, kann auch neue Zeichen definieren (z.B. so'n Quark wie ä, ö, ü oder gar ß). BASIC reicht dafür aber hinten und vorne nicht aus.

Die Pseudografik ist die einfachste Möglichkeit Zeichen und grafische Elemente auf einem Bildschirm zu mischen. Es gibt Striche, Winkel, Diagonalen und die Spielkartenfarben Pik, Herz, Karo, Kreuz (amerikanische Reihenfolge).

Diese Pseudografik-Zeichen bilden zusammen mit den übrigen Zeichen den ATARI-Zeichensatz. Jedes Zeichen hat eine Code-Zahl zwischen 0 und 255, den ATASCII-Code (vgl. 1.2.2).

Statt ein bestimmtes Zeichen über die Tastatur einzutippen, kann man es auch über seinen ATASCII-Code aufrufen. Das A hat den ATASCII-Wert 65. Der Befehl PRINT "A" bewirkt das selbe wie der Befehel PRINT CHR$(65). Der Wert 65 wird durch CHR$ in das entsprechende ATASCII-Zeichen umgewandelt.

Warum ich das hier so ausgiebig erkläre? Ich möchte Ihnen gleich ein kleines Programm in Pseudografik vorstellen, komme da aber in Bedrängnis, denn die Drucker verfügen zwar alle über den standardisierten ASCII-Zeichensatz, nicht aber über die Semigrafik-Zeichen. Deshalb finden Sie in Zeile 30 den Befehl CHR$. Hier nun das Programm. Ein kleiner Heimatgruß an unsere bayerischen Leser:

```
10 POKE 710,98:POKE 82,0
20 FOR I=0 TO 478
30 ? CHR$(8);CHR$(136);
40 NEXT I
50 GOTO 50
```

10: ein Farbwert wird gePOKEt und - ach lassen Sie mir noch ein paar kleine Geheimnisse für die restlichen Kapitel des Buches.

20: Na und das kennen Sie ja nun schon. Wie die Zahl 478 zustande kommt, erkläre ich Ihnen in Zeile

30: Die beiden CHR$s rufen zwei Pseudografik-Zeichen auf. Sie können die Zeichen aber auch als String schreiben. Dazu müssen Sie folgendes eintippen: ? " Taste CONTROL und H gleichzeitig, dann die ATARI-Taste (bei den alten Geräten war das ATARI-Zeichen auf dieser Taste, das nennt man Produktimage, bei den neueren Geräten ist ein diagonal geteiltes Rechteck, halb hell, halb dunkel darauf. Nach dem Drücken dieser Taste werden alle Zeichen invers, also negativ ausgegeben; wird durch erneutes Drücken rückgängig gemacht.) und Tasten CONTROL und H, ATARI- (INVERS-) Taste " und das ; nicht vergessen. Der zu druckende String zwischen den " " ist also zwei Zeichen lang. Der Bildschirm hat 40*24 Schreibstellen = 960. Da bei jedem Durchlauf zwei Zeichen gedruckt werden, brauchen wir 480 (0 bis 479) Durchläufe, bis die Mattscheibe voll ist. Um für den Cursor Platz zu lassen, begnügen wir uns mit 478. Ändern Sie den Schleifenzähler auf 479 und sehen Sie selbst.

40: die NEXT-Markierung der Schleife I

50: hält das Programm in dieser Zeile fest, damit der Rechner nicht die Arbeit niederlegt, sich mit READY meldet und damit unser schönes Bayern-Bild befleckt.

1.4.5 POS.itionen

Mit dem ?-Befehl werden die Zeichen eines Strings oder die Werte numerischer Variablen an den Bildschirm ausgegeben. Die Ausgabe beginnt oben links in der Ecke und erfolgt fortlaufend wie beim Schreiben auch. Da es (in Amerika) TV-Geräte gibt, die mit dem Bild knausern, ist in der Grundeinstellung die Spalte 2 als linker Rand eingestellt. (Mit POKE 82,n können Sie den linken, mit POKE 83,m den rechten Rand verändern.)

Ist die Ausgabe auf den Bildschirm gePRINTet, springt der Cursor sofort an den Anfang der nächsten Zeile. Es sei denn, wir befehlen ihm mit ; stehenzubleiben oder aber mit , zum nächsten TAB-Stop zu springen. Um die Silbentrennung schert sich das Elektronengehirn natrülich nicht. Ist eine Zeile voll, gehts in der nächsten weiter. Basta.

Soll ein Text nicht am linken Rand beginnen, kann man den String mit entsprechend vielen Leerzeichen beginnen lassen: " Platz".

Es gibt aber auch die Möglichkeit, eine ganz bestimmte Position des Bildschirms anzusprechen und ein Zeichen hineinzuschreiben. Dazu ist der Bildschirm horizontal, also in X-Richtung in 40 Spalten (0 bis 39) und vertikal also in Y-Richtung in 24 Zeilen (0 bis 23) aufgeteilt. Auf diese Weise kann mit zwei Koordinatenwerten jede einzelne Stelle der Mattscheibe erreicht werden.

Das Kommando dafür lautet POSITION n,m. POSITION darf mit POS. abgekürzt werden, n und m können numerische Werte (POS. 4,12), Variablen (POS. N,M) oder zu berechnende Werte (POS. X*3,Z+Y) sein. Ein POS.-Befehl:

```
20 POSITION X,Y
```

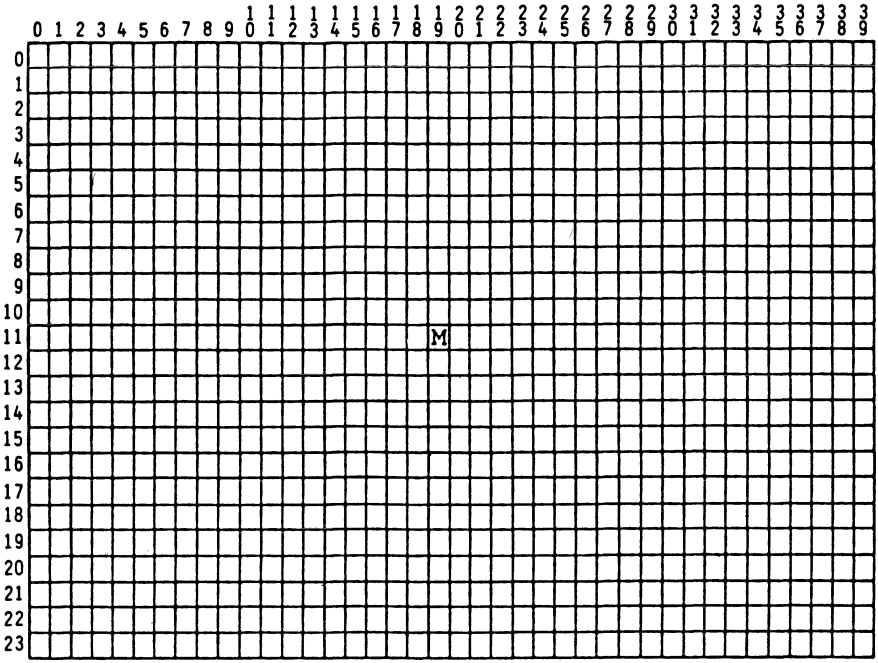
wird aber erst wirksam, wenn ein Ausgabebefehl erfolgt:

```
30 ? "M"
```

Wenn wir X und Y jetzt noch mit Werten ausstatten:

```
10 X=19:Y=11
```

können wir das Programm laufen lassen: RUN



Natürlich können Sie in diesem Fall statt der Variablen auch gleich die Werte einsetzen:

```
40 POSITION 19,11:? "M"
```

Eine POS.itionierung mit unzulässigen Werten allein führt noch nicht zu einer Fehlermeldung:

```
100 POSITION 19,25    (und RETURN-Taste)
```

wenn Sie das Programm aber RUNnen, erscheint nichts als:

```
ERROR-    141
```

auf der Mattscheibe (Cursorposition unzulässig), denn eine Zeile 25 kann der arme Rechner einfach nicht finden.

Und wozu ist POS. nun gut? Zum Beispiel, wenn Sie einen Zähler auf den Bildschirm bringen wollen, der an einer bestimmten Stelle steht, vielleicht in der Mitte der Mattscheibe und als Anzeige mit wechselnden Ziffern zählt (die Zeit, Punkte in einem Spiel oder was weiß ich):

```
10 FOR I=0 TO 9
20 POSITION 19,11
30 ? I
40 NEXT I
```

RUN und zack! Das geht wohl etwas zu schnell und der Cursor macht sich auch nicht so gut. Also erstmal der Cursor unterdrückt:

```
5 POKE 752,1
```

(POKE 752,0 macht ihn wieder sichtbar) und dann noch:

```
60 GOTO 10
```

damit wir länger etwas von dem Spaß haben. Jetzt müssen wir den Arbeitseifer des Rechners noch ein wenig dämpfen, denn so sind die Ziffern ja gar nicht zu erkennen. Und wie macht man das (nicht nur mit Beamten und lästigen Mitarbeitern?), wir geben ihm etwas zu tun:

```
45 FOR Z=0 TO 100:NEXT Z
RUN
```

Ah, ja, so ist es schön. Aber ein Zähler hat eigentlich ein paar Stellen mehr, oder? Was, kein Problem? Ja dann ändern Sie mal Zeile:

```
10 FOR I=0 TO 1000
```

und sehen sich selbst an, was dabei herauskommt. Nein, wenn die einzelnen Stellen des Zählers immer schön an der gleichen Stelle bleiben sollen, dann müssen wir uns schon etwas mehr Mühe geben:

```
5 POKE 752,1
10 FOR I=0 TO 9
20 POSITION 17,11:? I
30 FOR J=0 TO 9
40 POSITION 18,11:? J
50 FOR K=0 TO 9
60 POSITION 19,11:? K
70 FOR L=0 TO 9
80 POSITION 20,11:? L
90 FOR M=0 TO 9
100 POSITION 21,11:? M
110 NEXT M
120 NEXT L
130 NEXT K
140 NEXT J
150 NEXT I
160 GOTO 10
```

Das ist vom Aufbau her das gleiche Programm wie der einstellige Zähler, nur daß wir für jede Stelle dieses Zählers eine FOR-NEXT-Schleife haben, also für fünf Zählstellen fünf Schleifen: eine Schleife in einer Schleife in einer Schleife in

einer Schleife in einer Schleife. Die innerste Schleife zählt die Einer. Wenn Sie von 0 bis 9 durchgelaufen ist, wird die nächst größere Schleife (für die Zehner) angesprungen und dann wird wieder die innerste Schleife völlig abgearbeitet usw.

Wenn die Zahl 99999 auf dem Bildschirm erscheint, sind alle fünf Schleifen abgearbeitet und das Programm kommt endlich nach Zeile 160, wo es wieder nach 10 zurückgeschickt wird (mit einem Computer kann man das ja machen) und die Zählerei wieder mit 00000 beginnt.

Der besondere Hobby-Tip:

wie macht sich ein Computer-Besitzer bei seiner Familie beliebt? Erweitern Sie das Zähler-Programm auf, sagen wir 30 Stellen (eine schöne Fingerübung) und schreiben Sie in der letzten Zeile statt eines Rücksprunges:

```
910 ? "Fertig! Ihr koennt jetzt wieder Fernsehen!"
```

(Wenn Ihre Familie keinen Spaß versteht, können Sie auch einen Wettlauf veranstalten. Starten Sie das Programm, wenn Sie ins Bett gehen, und schauen Sie gleich nach dem Aufstehen nach, wer schneller war.)

1.4.6 Komm, PLOT auf den Bildschirm

Was dem String das POS. ist dem Grafikpunkt das PLOT.

Der Befehl POS. ist nur in GR.0 wirksam. Auch wenn eine Grafikart mit Textfenster gewählt wurde, kann zwar dorthin ge?tet werden, aber der POS.-Befehl zieht nicht.

In den Textbetriebsarten GRAPHICS 1, 2, 12 und 13 kann mit POS. eine BildschirmPOSition zwar angesteuert werden, aber der PRINT-Befehl muß so erweitert werden, daß das Betriebssystem erkennen kann, wohin gePRINTet werden soll. Um in diesen Betriebsarten ins Grafikfenster zu PRINTen, muß ein Datenkanal zum Grafikfenster (was immer das auch sein mag) angesprochen werden. Der Befehl sieht dann so aus:

```
10 GRAPHICS 1:POSITION 4,16:? #6;"TEXT"
```

(mehr dazu in den Abschnitten 2.2.1 und 2.2.2)

Wärend der PRINT-Befehl Strings, also definierte Zeichen aus dem ATARI-Zeichensatz in die Schreibstellen setzt, werden mit PLOT die Bildschirmpositionen bestimmt, in die Grafikpunkte gedruckt werden sollen. Die Form des Grafikpunktes ist abhängig von der Auflösung der gewählten Betriebsart, aber ein Grafikpunkt füllt eine solche Schreibposition vollständig aus. Die Farben für die Grafikpunkte werden mit SETCOLOR (oder POKE) definiert und mit COLOR aufgerufen:

```
10 GRAPHICS 3:POKE  (beliebige Farbwerte einsetzen)
20 COLOR 1:PLOT 0,0
30 COLOR 2:PLOT 0,1:PLOT 1,1
RUN
```

Wenn Sie brav waren und wirklich GR.3 programmiert haben, dann sehen Sie jetzt ein Quadrat in der von Ihnen gewählten Farbe 1 und darunter ein Rechteck in Farbe 2. Wenn Sie aber das Textfenster ausgeschaltet haben (GR.3+16), dann sehen Sie jetzt gar nichts, denn wenn ein Programm abgearbeitet ist, meldet sich der Rechner mit READY. Das muß er irgendwohin schreiben. Wenn Sie ihm dafür nicht einmal das kleine Textfenster belassen haben, dann knipst er eben GR.0

nur um sein blödes READY loszuwerden. Um ihm das auszutreiben, nageln wir ihn in einer Zeile fest:

```
40 GOTO 40
```

Die mit COLOR (Abkürzung CO., XL-Reihe C.) eingeschaltete Farbe wird so lange gePLOTtet, bis mit dem nächsten COLOR die Farbe gewechselt wird.

Wenn nun als nächstes ein Balken gezogen werden soll, dann können sie natürlich jeden einzelnen Punkt des Balkens PLOTten:

```
35 COLOR 3:PLOT 0,2:PLOT 1,2:PLOT    ...    :PLOT 11,2
```

Ich lasse da lieber eine FOR-NEXT-Schleife für mich arbeiten:

```
35 COLOR 3:FOR G=0 TO 11:PLOT G,2:NEXT G
```

Aber der gute ATARI verwöhnt uns mit einem noch viel komfortableren Befehl und der heißt: ZIEHLEINE (natürlich auf Englisch: DRAWTO, was man DR. abkürzen darf). Und deshalb sollten wir lieber eintasten:

```
35 COLOR 3:PLOT 0,2:DRAWTO 11,2
```

Der DRAWTO-Befehl bewirkt, daß eine Linie von dem zuletzt gePLOTteten Punkt zu dem hinter DR. definierten Punkt gezogen wird. Das Angenehme dabei ist, daß beliebige zwei Punkte auf diese Weise verbunden werden können, also auch schräg liegende Geraden (um es mal ganz mathematisch auszudrücken). Bitten Sie Ihren Computer doch einmal um folgendes:

```
10 GRAPHICS 3:POKE    (beliebige Farbwerte)
20 COLOR 1:PLOT 0,0:DRAWTO 2,10
30 COLOR 2:PLOT 2,0:DRAWTO 6,10
40 COLOR 3:PLOT 4,0:DRAWTO 10,10
RUN
```

Zugegeben, die Diagonalen sehen etwas geknickt aus, aber die Auflösung ist ja auch sehr grob. Je höher der GRAPHICS-gang wird, desto gerader werden die Geraden.

Und eine zweite Wohltat hält DR. bereit. Es können beliebig viele DR.s aneinandergehängt werden, so daß eine durchgehende Linie entsteht, die an bestimmten Stellen die Richtung wechselt:

```
10 GRAPHICS 3
20 COLOR 2
30 PLOT 19,0:DRAWTO 28,9:DRAWTO 19,18
40 DRAWTO 10,9
RUN
```

Na is' doch klar. Der Rechner tut genau das, was wir ihm programmieren. Wenn also ein Quadrat erscheinen soll, dann muß auch noch von der vierten Ecke zur ersten eine Linie gezogen werden:

```
50 DRAWTO 19,0
```

So, mit der Basis-BASIC-Paukerei sind wir nun durch. Wurde aber auch Zeit, was. Zum Abschluß und als ersten Vorgeschmack auf kommende Freuden, noch eine kleine Erweiterung des obigen Prograämmchens:

```
10 GRAPHICS 3
20 FOR C=1 TO 3:COLOR C
30 PLOT 19+C,0:DRAWTO 28+C,9:DRAWTO 19+C,18
40 DRAWTO 10+C,9:DRAWTO 19+C,0
50 NEXT C
```

Ach ist das schön!

10: bestimmt den GR.afikgang

20: eröffnet die Schleife mit der Variablen C Wert 1

30: Beim ersten Durchlauf ist C also 1 und mit COLOR C wird Farbe 1 aufgerufen. Zu allen X-Koordinaten von PLOT und DRAWTO wird C also 1 addiert, das entspricht einer Verschiebung um einen Grafikpunkt/Schreibstelle nach rechts. Beim nächsten Durchlauf wird das gleiche Quadrat gePLOTtet, nur mit Farbe 2 und um zwei Stellen nach rechts verschoben (C=2). Beim dritten Druchlauf geschieht entsprechendes mit Farbe 3.

50: dann ist die Schleife abgearbeitet.

Beachten Sie bitte,
daß bei der oberen und unteren Ecke des Quadrates die Farbe 3 je einen Grafikpunkt der Farbe 1 überPLOTtet. Ein gePLOTteter Grafikpunkt bleibt so lange stehen, bis er überPLOTtet wird, mit einer anderen Farbe. Und wenn das die Farbe des Hintergrundes ist, dann sieht das so aus, als ob der Punkt verschwinde. In Wirklichkeit ist das Grafikfenster lückenlos mit Grafikpunkten ausgefüllt. Wir nehmen nur die durchgehende Flächer als Hintergrund wahr und den Hintergrund stellen wir uns als leeren Raum vor. Kleinere Flächen deutet unser Gehirn-Computer dann als "Gegenstände" (Vordergrund). Wahrnehmung ist also nur die Annahme, daß etwas wahr sei und kein Wahrheitsbeweis. Deswegen läßt sich die Wahrnehmung auch gezielt täuschen. Die "bewegten" Bilder des Kinos sind nur ein Beispiel dafür. Auch wir werden der Wahrnehmung noch einiges vorgaukeln.

1.4.7 Fill-Horn

Es gibt Heim-Computer, die verfügen über einen Befehl, mit dessen Hilfe eine definierte Fläche farbig ausgefüllt (FILL) werden kann. Der ATARI gehört leider nicht dazu. Es gibt aber einen Weg, diesen Effekt zu bewirken. Das ATARI-Zauberwort dafür heißt XIO.

Folgende Schritte sind nötig, um eine Fläche farbig auszufüllen: In die Adresse 765 muß vermittels POKE der Farbwert geschrieben werden, mit dem die Fläche gefüllt werden soll. Dann müssen die rechte und obere Kante des auszufüllenden Vierecks mit PLOT und DRAWTO gezeichnet und schließlich muß der unsichtbare Cursor auf die untere linke Ecke der Fläche POS.itioniert werden. Dann erst kann das XIO-Kommando ergehen:

```
100 GRAPHICS 23
110 POKE 765,2
120 COLOR 2
130 PLOT 159,79
140 DRAWTO 80,0
150 DRAWTO 79,0
160 POSITION 0,79
170 XIO 18,#6,0,0,"S:"
180 GOTO 180
```

100: schaltet die fein auflösende Grafikbetriebsart 7 ohne Textfenster (+16=23) ein.

110: bestimmt für das Ausfüllen die Farbe (COLOR) 2

120: schaltet COLOR 2 für PLOT und DRAWTO ein,

130: PLOTet die Ecke unten rechts,

140: zieht eine Linie nach oben rechts,

150: führt die Linie nach oben links weiter und

160: POS.itioniert den unsichtbaren Cursor auf die Ecke unten links.

170: gibt den Befehl zu Ausfüllen der Fläche. Das Nummernzeichen (auf deutsch "Raute" geheißen) bezeichnet einen Datenkanal mit der Nummer 6, der nach "S:", dem Grafik-

fenster (Screen) eröffnet werden soll. Aber das braucht uns hier alles nicht zu kümmern. Achten Sie nur darauf, den Befehl ohne Tippfehler hinzukriegen.

180: verhindert das Programmende mit READY.

RUN

Ach! Das ist ja doch ein Dreieck! Weit gefehlt. Es kann grundsätzlich nur ein Viereck gefüllt werden. Was sich hier wie ein Dreieck gibt, ist in Wirklichkeit ein Trapez. Die obere Kante dieses Vierecks ist jedoch nur zwei Grafikpunkte lang (kürzer gehts ja nicht) und das sieht dann wie ein Dreieck aus. Ja unsere Wahr-nehmung.

Ändern Sie nun in Zeile 110 den POKE in 763,3. Die Fläche wird jetzt in Blau ausgefüllt und Sie können besser beobachten, wie das geschieht. Erst werden die linke und obere Kante mit COLOR 2 gezogen. Dann wird die linke Kante Punkt für Punkt gePLOTtet und nach jedem PLOT wird die Zeile bis zur rechten Kante mit COLOR 3 (POKE 765,3) gefüllt, bis der mit POS. bestimmte Punkt erreicht ist. Die untere Kante des Vierecks hat also die Füll-Farbe. Wenn Sie auch eine durchgehende Umrandung schöner finden, dann löschen Sie Zeile 180 und ergänzen:

```
190 PLOT 0,79
200 DRAWTO 159,79
210 GOTO 210
```

Und weil wir gerade dabei sind, sollten Sie sich noch Zeit für folgendes Experiment nehmen. Der Bildschirm ist jetzt zur Hälfte von einem blauen Dreieck bedeckt, das eine gelbe Umrandung hat. PLOTten wir doch als nächstes ein Rechteck, von dem das ganze Grafikfenster eingefaßt wird und füllen dieses Grafikfenster aus. Mal sehn, was passiert:

```
210 löschen!
220 PLOT 159,79
230 DRAWTO 159,0
240 DRAWTO 0,0
250 POSITION 0,79
260 POKE 708,50
270 POKE 765,1
280 XIO 18,#6,0,0,"S:"
290 GOTO 290
```

Und was sehen Sie? Halbe Arbeit! Mit XIO ausgemalte Flächen können nicht erneut gefüllt werden, bereits gePLOTtete Linien erkennt XIO als rechte Begrenzung auf seinem Füllweg.

1.5 Hilfe!

In diesem Kapitel sind die schmerzlichen Erfahrungen durchprogrammierter Nächte versammelt, Wunden aus verbissenen Kämpfen mit Programmen, die einfach nicht laufen wollten, mit Fehlern, die nicht zu entdecken waren, mit falschen Anleitungen und unrichtigen Hinweisen.

Außerdem will ich an dieser Stelle ein paar Tips einstreuen, auf die sonst verzichtet wird, weil sie so banal klingen (und welcher Autor möchte schon banal wirken), für die ich als Anfänger aber dankbar gewesen wäre und die mir manchen Frust erspart hätten.

Wenn Sie eine BASIC-Zeile mit einem Fehler eingegeben haben (RETURN-Taste), erscheint ein ERROR- gefolgt von der fehlerhaften Zeile und ein Cursor steht auf (dem Tippfehler) oder vor der fehlerhaften Anweisung. Überschreibt die Fehlermeldung möglicherweise eine andere BASIC-Zeile, so ist diese deshalb nicht verloren! Sie ist noch im Speicher und auf das Kommando LIST ist sie wieder da.

Sehen Sie sich also die fehlerhafte Zeile gut an. Wenn Sie den Fehler entdeckt haben, fahren Sie den Cursor nach oben und setzen ihn auf die Zeile mit der ERROR- Meldung. Drücken Sie die Tasten SHIFT und DELETE gleichzeitig und die Zeile verschwindet. Gehen Sie jetzt weiter nach oben und berichtigen den Fehler. Sie können also in den Programmzeilen auf dem Bildschirm beliebig herumfahren und Änderungen vornehmen. Eine geänderte Zeile müssen Sie aber immer durch Druck auf die RETURN-Taste verlassen. Der Rechner nimmt dann die korrigierte Zeile als neue Zeile an.

Wenn Sie eine fehlerhafte Zeile nicht sofort berichtigen, müssen Sie später die ganze Zeile neu schreiben. Der Rechner speichert nämlich nur die Zeile, die mit ERROR- beginnt und die läßt sich nicht verändern. Es ist also nicht möglich, das "ERROR- " zu löschen und den Fehler zu korrigieren.

Auch sonst merkt sich der Computer bei Zeilen mit gleichen Nummern immer die zuletzt eingegebene. Damit ist das Löschen einer BASIC-Zeile möglich. Tippen Sie die Zeilennummer und RETURN-Taste. Die alte, zu löschende Zeile wird durch diese leere Zeile ersetzt und leere Zeilen werden vom

Betriebssystem ignoriert; also: Zeile weg.

Abgesehen von Tippfehlern machen sich die meisten Fehler erst bemerkbar, wenn das Progamm läuft, oder vielmehr laufen soll. Statt wie besessen draufloszutippen, sollten Sie Abschnitte Ihrer Programm-Kreation zwischendruch testen und wenn Sie ein Speichermedium (Cassette oder Diskette) haben auch abspeichern. Erst wenn ein Programmteil fehlerfrei läuft, sollten Sie den nächsten Abschnitt in Angriff nehmen.

Das Kommando LIST bringt das LISTing, also die Folge der BASIC-Zeilen (nach Zeilennummern geordnet) auf den Bildschirm (und erweitert auch auf Drucker, Cassette oder Diskette). Mit LIST Zeilennummer,Zeilennummer können Sie aus dem Gesamtprogramm auch nur ein paar Zeilen oder eine einzelne Zeile (LIST Zeilennummer) ausdrucken lassen.

Die meisten Programme sind länger als das Fassungsvermögen des Bildschirms. LIST (und RETURN) startet die Ausgabe, drücken Sie jetzt die Tasten CONTROL und 1, wird das Aus-LISTen unterbrochen, und Sie können in Ruhe nach Fehlern suchen. Erneutes Drücken von CONTORL und 1 setzt das LISTen fort.

Wenn Sie das LISTing unterbrochen und einen Fehler gefunden haben, können Sie natürlich mit dem Cursor hinfahren, den Fehler beheben und mit RETURN abspeichern. Dabei ist aber Vorsicht geboten. Bei Druck auf CONTORL und 1, wird das LISTing irgendwo, wahrscheinlich mitten in einer BASIC-Zeile unterbrochen. Die letzte Zeile auf dem Bildschirm ist also eine verstümmelte BASIC-Zeile. Nun ist es sehr verlockend, nachdem die korrigierte Zeile mit RETURN verlassen wurde einfach so oft auf die RETURN-Taste zu drücken, bis der Cursor wieder am unteren Rand des Bildschirms ist. Dabei würde aber die letzte Zeile in ihrer verstümmelten Form abgespeichert.

Das Bearbeiten eines Programmes nennt man Editieren. Die Tastatur umfaßt einige Tasten, die nur für diesen Zweck da sind. Das sind zum einen die Cursor-Tasten und dann noch SHIFT, CONTORL, DELETE, INSERT, BACK-S, CLEAR.

CONTROL oder SHIFT und CLEAR
löscht den gesamten Bildschirm (nicht das Programm) und setzt den Cursor auf die Home-Position, in die linke obere Ecke also. Ihr wertvolles Programm bleibt also erhalten. Ein kleines LIST und Sie haben es wieder Weiß auf Hellblau vor Augen.

BACK-S(pace)
bewegt den Cursor um eine Schreibposition zurück und löscht

das dort stehende Zeichen. In Vorwärtsrichtung wird durch Betätigen der Leertaste gelöscht.

CONTROL und INSERT
schafft hinter dem Cursor eine freie Schreibstelle. Die übrigen Zeichen der BASIC-Zeile werden entsprechend verschoben.

SHIFT und INSERT
schafft dort wo der Cursor steht eine leere Zeile. Vorsicht. Wird eine entsprechend lange BASIC-Zeile dadurch zerhackt, bleibt wie beim Regenwurm nur das Kopfstück mit der Zeilennummer erhalten.

CONTROL und DELETE
löscht das Zeichen unter dem Cursor

SHIFT und DELET
löscht die Zeile, in der sich der Cursor befindet. Eine BASIC-Zeile wird ganz gelöscht, auch wenn sie mehrere Bildschirmzeilen lang ist. Die BASIC-Zeile ist jedoch nur auf dem Bildschirm gelöscht, nicht aber im Memory (also soviel Englisch müssen Sie als Computer-Novize schon verkraften)

Auf Seite 160 finden Sie ein kleines Programm, mit dem Sie diese Funktionen lernen können.

BREAK
unterbricht ein laufendes Programm. Wenn Sie beim Eingeben sind, können Sie durch Drücken der BREAK-Taste eine Zeile verlassen, ohne daß das gerade getippte Kommando auch ausgeführt wird. Die BREAK-Taste wirk wie die RETURN-Taste, der Cursor springt an den Anfang der nächsten Zeile, aber da kein EOL erfolgte (End Of Line, haben Sie das denn schon wieder vergessen? Seite 12), sieht sich der Computer auch nicht veranlaßt, irgendetwas zu tun.

Wenn Sie ein laufendes Programm unterbrechen erscheint die Meldung: STOPPED AT LINE (Zeilennummer). Daraus lassen sich manchmal Rückschlüsse auf Programmfehler ziehen. Ein unterbrochenes Programm können Sie weiterlaufenlassen, wenn Sie CONT (inue) eintippen.

(SYSTEM) RESET
ist die zweite Möglichkeit, ein Programm abzubrechen. Das Betätigen dieser Taste löst einen sogenannten Warmstart aus, d.h. der Computer wird in den Zustand nach dem Einschalten versetzt, nur daß das Programm im Speicher erhalten bleibt. Unter Kaltstart versteht man entsprechend das Einschalten nachdem zuvor die Stromzufuhr unterbrochen war, und das überlebt kein Programm im Speicher.

Und sicher wird es auch Ihnen nicht erspart bleiben, daß eines schönen Programmiertages plötzlich nichts mehr geht. Welche Taste Sie auch drücken. Nichts. Der Computer ist abgestürzt. Keine Angst, solange die Maschine nicht vom Tisch stürzt, passiert ihr nichts, nur wenn Sie abgestürzt ist, gibt es nur einen einzigen Weg: ausschalten, Programm ade! (Es hilft nichts und wenn das Programm noch so lang ist, das Sie gerade im Speicher haben)

Der Computer kann abstürzen, weil er zu warm geworden ist.

Der Computer kann abstürzen, weil mit irgendwelchen direkten Speichereingriffen Schaltungen veranlaßt wurden, die ihn in den Wahnsinn treiben.

Der Computer kann abstürzen, weil der Chip mal aushakt. Nur in der Rüstungstechnik werden Chips aller erster Wahl verarbeitet (klar doch), bei Heim-Computern rutscht schon mal ein beschwipster Chip durch. Wenn Ihre Kiste also "ohne Grund" dauern aufgibt, dann sollten Sie doch einmal ein freundschaftliches Wort mit Ihrem Fachhändler wechseln.

Also selbst, wenn Sie immer hübsch auf dem Pfad der BASIC-Tugend wandeln, kommt es hier und da einfach vor, daß Ihr ATARI streikt. Wenn man also seit einigen Nächten an einem Riesenprogramm herumhackt, dann sollte man immermal zwischendurch das Erreichte auf Band oder Diskette für den Fall der Fälle abspeichern.

Auch Zeilennummern lassen sich ändern! Fahren Sie mit dem Cursor in die entsprechende Zeile und ändern Sie die Nummer. Mit RETURN abspeichern. Natürlich denkt sich der Rechner nicht von selbst, daß das eine Änderung sein soll. Er nimmt die Eingabe als neue BASIC-Zeile in den Speicher auf. Sie müssen also ggf. die alte Zeile löschen.

Diese Erkenntnis läßt sich trefflich nutzen, wenn Sie in einem Programm mehrere ähnliche Zeilen haben. Statt haufenweise BASIC-Zeilen einzutasten, die sich nur in wenigen Zeichen unterscheiden, schreiben Sie die erste Zeile, fahren dann mit dem Cursor zurück, ändern die Zeilennummer und die abweichenden Zeichen, RETURN, schon ist die Zeile im Speicher. Auf diese Weise erhalten Sie nullkommanix jede Menge BASIC-Zeilen. Auf LIST erscheinen sie auch alle auf dem Bildschirm.

Für die Fehlersuche gibt es wenig Rat. Meist sind es ja logische Fehler, an denen das Programm scheitert. Man gewöhnt sich aber mit der Zeit daran, daß der Rechner ganz stur das tut, was man eintippt und nicht das, was man

sich dabei gedacht hat. Es hat also gar keinen Zweck zu denken, das Programm müsse so laufen, wie es geschrieben wurde. Wenn es nicht läuft, steckt irgendwo ein Fehler, auch wenn es manchmal schwerfällt, das zu akzeptieren.

Gehen Sie das Programm Zeile um Zeile durch, verfolgen Sie jede Verzweigung, jeden Rücksprung, überprüfen Sie jede Bedingung. Verfolgen Sie eine einzelne Variable und merken sich, welchen Wert sie jeweils hat, wenn im Programm mit ihr gearbeitet wird.

Auf jeden Fall lohnt es sich nach Tippfehlern zu suchen. Vielleicht steht an irgendeiner Stelle eine falsche Variable (H2 statt H3 oder gar HO statt H0). Auch die Richtigkeit der Satzzeichen , ; : lohnt es immer zu überprüfen. Und denken Sie immer daran, ein Programm zu schreiben ist die halbe Arbeit, die Fehler zu suchen die andere Hälfte.

So! Da dieses Buch nicht den Anspruch erhebt, alles über die Bedienung des ATARI zu vermitteln oder ein erschöpfender BASIC-Kurs zu sein, soll an dieser Stelle endlich Schluß sein mit der Paukerei.

2 Bildschirmbilder

Was wäre der Computer ohne Bildschirm. Und was wäre der Bildschirm ohne Farbe. Die gängigen Heim-Computer üben sich heute alle in Schönfärberei. Der ATARI ist Primus, wenns um Grafik und Farbe geht. Auf welchen geheimnisvollen Pfaden die Farbe vom Rechner angerührt wird, wie sie dann in den Fernsehkasten fließt und dort die Mattscheibe bunt färbt, das soll in diesem Abschnitt ahnungsvoll erzählt werden.

2.1.1 Lange Leitung

Daß die Bilder hinten an Ihrem Computer aus einer kleinen runden Steckdose herauskommen und von dort vermittels eines passenden Kabels durch den Antenneneingang in den Fernseher fließen, haben inzwischen außer Ihnen auch die übrigen Familienmitglieder gemerkt (weil das Fernsehprogamm in letzter Zeit dauern ausfällt).

Der Computer sendet also Signale, die vom Fernsehgerät empfangen werden wie der Freitagskrimi. Je nach Geräteverträglichkeit und Verlauf des Übertragungskabels können geringe Bildstörungen auftreten, ein leichtes Schwimmen oder ein an die Bildhelligkeit gekoppelter Brummton. Beim ATARI halten sich diese Mängel in Grenzen.

Weil so ein Fernsehbild aber immer etwas unscharf ist und Ihre Familie auch mal wieder eine Quiz-Sendung sehen will, sollten Sie sich bei Zeiten einen Monitor zulegen.

Über den Antennenausgang können Sie auch eine Verbindung zu einen Videorecorder herstellen. Sie können dann aufzeichnen, was der Computer auf dem Bildschirm anstellt. So läßt sich mit einem Programm, das eine farbige Grafik Schritt für Schritt aufbaut und/oder verändert (Beispiele finden Sie im nächsten Abschnitt) der Vorspann für einen selbstgedrehten Videofilm oder ein vollständiger Trickfilm auf die Mattscheibe und die Videocassette bannen.

Es ist allerdings nicht ganz problemlos, den Videorecorder sauber auf den Computer einzustellen. Wegen der Gerätevielfalt läßt sich hierzu nichts verbindliches sagen. Es ist jedoch auf jeden Fall vorteilhaft, wenn die Geräte (TV, Computer und Recorder) nicht zu dicht zusammenstehen. Alle Geräte haben ein elektromagnetisches Feld. Dadurch können die Bildsignale gestört werden. Hier sollten Sie auch besonders auf die Netzgeräte achten.

Wenn Sie aber erst eine saubere Einstellung ausgetüftelt haben, werden Sie als Videofilmer immer neue Einsatzmöglichkeiten für den Heim-Computer finden.

2.1.2 Mattscheibe

Erst seit der Computer ein gutes Bild abgibt, ist er salonfähig geworden. Aber was für den User (das sind Sie) so problemlos aussieht: Kabel in den Antenneneingang, Computer an, TV an, Bild da - ist in Wirklichkeit ein delikater Prozeß, der vielerlei Probleme aufwirft.

Bekanntlich besteht das Fernsehbild aus einer großen Menge kleiner Punkte, die in Zeilen angeordnet sind. Ein durch Magnetfelder abgelenkter Kathodenstrahl tastet den Bildschirm Punkt für Punkt ab. Er beginnt in der Ecke oben links, wo sonst, und schreibt eine Zeile. Am Ende Zeile schaltet er kurz ab (horizontal blank) und springt an den Anfang der nächsten Zeile und so fort, bis er unter rechts angekommen ist. Dort schaltet er für etwas länger ab (vertical blank), um wieder nach oben links zu springen.

Die Signale, die der Fernsehapparat bekommt, müssen mit diesen Arbeitstakten genau übereinstimmen, sonst fängt das Bild an zu laufen oder es ist überhaupt nichts zu erkennen. Der Computer muß also mit dem TV-Gerät exakt synchronisiert sein. Für diese Arbeit ist der ATARI mit einem extra Chip ausgestattet. Das erklärt auch die brillierenden Grafikmöglichkeiten und den Farbenreichtum. Dieses Bauelement heißt ANTIC. Es liest die Display-list, die angibt, wie die Bildschirmzeilen des Fernsehers für die Grafikdarstellung aufgeteilt werden sollen.

Theoretisch könnte natürlich jeder Bildschirmpunkt gesondert angesprochen werden. Das würde aber eine Speicherkapazität fordern, die (noch) kein Heim-Computer zu bieten vermag. In den verschiedenen Grafikbetriebsarten werden deshalb immer Gruppen von Punkten informationsmäßig zusammengefaßt.

ANTIC holt aus dem Bildschirmspeicher die Informationen, die das Bild enthalten und übergibt sie an einen Baustein (CTIA), der sie in Videosignale umsetzt. Von den 262 Bildschirmzeilen, die von ANTIC verwaltet werden, sind aus technischen Gründen nur etwa 200 sichtbar, und damit es auch bei exotischen Fernsehgeräten keine Probleme gibt, werden in den vorprogrammierten Display-Lists (GRAPHICS 0

bis 15) nur 192 Bildschirmzeilen tatsächlich auf der Mattscheibe zum Leuchten gebracht. Wenn in GR.0 z.B. 24 Textzeilen dargestellt werden und jede dieser Zeilen acht Bildschirmzeilen hoch ist, so ergibt das (24 mal 8?) 192.

Diese 192 Bildschirmzeilen sind in jeder Grafikbetriebsart anders aufgeteilt. In GR.8 gibt es z.B. 192 Zeilen mit je 320 Bildpunkten, d.h. jeder einzelne der 61.440 Bildpunkte (Pixel), die der Computer von sich gibt, kann für sich angesprochen werden. Das kostet viel Speicherplatz und deshalb gibt es in GR.8 auch nur eine Farbwahlmöglichkeit.

Mit etwas Wissen über die Funktionsweise von ANTIC und mit einer Landkarte für die Gehirnzellen des Computers kann man die vorprogrammierten Display-lists durch eine selbst komponierte zusammenstellen. Auf diese Weise ist es möglich, verschiedene Grafikbetriebsarten beliebig auf dem Bildschir (zeilenweise) zu mischen, wenn nur die Gesamtzahl der Bildschirmzeilen 192 ergibt.

So lassen sich denn auch oberhalb des gewöhnlich vom ATARI dargestellten Bildschirmbereichs (bei normalen deutschen TV-Geräten) bis zu 32 Bildschirmzeilen, also z.B. vier GRAPHICS-0-Zeilen beschreiben.

Aber solche Eskapaden sparen wir uns doch wohl lieber für später auf, oder?

2.1.3 Schöner Schein

Da jeder zivilisierte Mensch heute einen Color-TV in der guten Stube stehen hat, wissen Sie sicher auch, daß all die schönen Farben aus nur drei Grundfraben zusammengemischt werden. Die Techniker sprechen da fröhlich von Rot, Grün und Blau. Na ja, wollen wir es ihnen nicht übel nehmen.

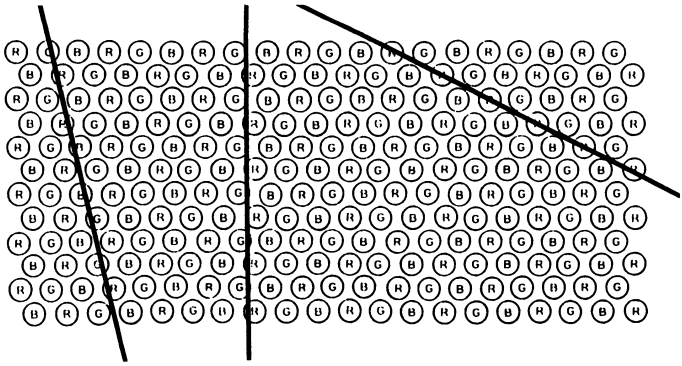
In der Schule haben Sie ja sicher schon einmal in ein paar Farbtöpfen gerührt und dabei staunend beobachtet, daß sich die Farben vermischen. Blau und Gelb mischen sich zu Grün. Aus diesem Mysterium leiten die stolzen Kunsterzieher mit überlegenem Lächeln ab, daß es nur drei Grundfarben gibt (Gelb, Rot, Blau oder richtiger Gelb, Magenta, Cyan), aus denen alle anderen Farbtöne gemischt werden können (womit sie ja auch Recht haben).

Das Mischen von Farben entspricht optisch dem Ausfiltern. Das "weiße" Licht eines Scheinwerfers wird durch ein blaues Filter blau gefärbt. Schieben wir auch noch ein gelbes Filter davor, wird das Licht grün. Wenn dann irgendein Tölpel auch noch ein rotes Filter vorschiebt, ist es stockdunkel. Diesen Vorgang nennt man subtraktive Farbmischung, weil vom Weiß einzelne Farbanteile ausgefiltert (subtrahiert) werden.

Beim Fernseher ist das ganz anders. Hier haben wir es mit farbigen Lichtquellen zu tun, den Bildschirmpunkten. Diese Farben strahlen zusammen, ihre Wellen addieren sich: additive Farbmischung. Und die ist nun wirklich recht eigenartig, oder besser, sie kommt uns so vor, weil wir diesen Vorgang im täglichen Leben kaum erfahren. Es ist nämlich tatsächlich so, daß sich orangenes Licht und grünes Licht zu gelbem Licht mischen! Orangenes und violettes Licht mischen sich zu rot, grünes und violettes zu Blau. Alle drei Scheinwerfer zusammen, Orange, Grün und Violett ergeben weißes Licht. Ob Sie's glauben oder nicht. Durch die Dosierung dieser drei Lichtquellen (farbtheoretisch richtig sind das also Orange, Grün und Violett) lassen sich alle Farbschattierungen erzeugen.

Ein Farbfernsehbildschirm besteht also aus Pünktchen, die, lassen wir den Technikern ihren Spaß: rot, grün oder blau

aufleuchten können und im Zusammenwirken die verschiedenen Farbtöne erzeugen.



Wenn nun in der hochauflösenden Grafik (HIRES) von GR.8 eine Linie gePLOTtet wird, so kann es sein, daß diese Linie nur bestimmte Farbpunkte berührt. Und obwohl in dieser Grafikart keine weiteren Farben gewählt werden können, werden Farbwirkungen auf dem Bildschirm sichtbar.

Versuchen sie folgendes kleine Programm, um sich dieses Phänomen anzuschauen:

```
10 GRAPHICS 8+16
20 POKE 710,0
30 FOR X=0 TO 319 STEP 10
40 COLOR 1
50 PLOT X,0
60 DRAWTO X,191
70 NEXT X
80 GOTO 80
```

10: wählt HIRES-Grafik ohne Textfenster (+16) und

20: POKEt als Hintergrundfarbe 0 = Schwarz, das bringt die Farben am besten zur Wirkung. Hier könnte auch stehen SETCOLOR 2,0,0

30: Der Schleifenzähler X läuft von 0 bis 319 in Zehnerschritten.

40: schaltet Farbe 1 an. Es gibt in diesem Grafikmodus zwar keine andere Farbe, aber eingeschaltet werden muß trotzdem, zur Wahl steht ja auch noch Farbe 0 (Hintergrund)

50: zeichnet eine Punkt an der oberen Bildschirmkante und zwar in den Spalten 0 bis 319 in Zehnerschritten und

60: zieht eine Linie zur entsprechenden Position am unteren Rand.

70: beendet die Schleife und

80: sorgt dafür, daß unser schönes Bild nicht für READY ausgeknipst wird.

Falls Sie dieses Programm wirklich laufen lassen sollten, werden Sie Ihr buntes Wunder erleben. Aber nicht genug damit! Ändern Sie Zeile:

```
60 DRAWTO 319-X,191
```

Das bewirkt, daß statt senkrechter Linien Diagonalen gezogen werden, die sich im Zentrum schneiden. Nun geben Sie schon ein RUN. Schillernde Farben erwarten Sie: Gelb, Lila, Grün und Purpur.

Bei diesem schönen Testbild tritt noch ein weiterer Effekt auf (deswegen sollten Sie nun wirklich endlich das kleine Programm oben eintippen, so viel Arbeit ist das doch gar nicht). Die gePLOTteten Linien setzen sich aus vielen Pixels zusammen und obwohl diese Punkte recht klein sind, ist die Auflösung für unser naturverwöhntes Auge doch recht grob. Eine schräg verlaufende Linie bekommt deutlich sichtbare Brüche. Das kann so weit gehen, daß sich eine Linie in einzelne Punkte auflöst. Unsere Wahrnehmung (über deren Wahrheitsgehalt ja schon früher nachgedacht wurde) faßt benachbarte Elemente zusammen. So kann es geschehen, daß Bündel von Linien durch den Punktraster der Mattscheibe zerbrochen werden und unser Gehirn die Bruchstücke zu neuen Formationen zusammenfaßt. Beim vorliegenden Programm ist das im Zentrum, wo sich alle Linien schneiden, deutlich zu sehen.

Eine ähnliche Wirkung entsteht, wenn zwei Lagen einer Gardiene übereinanderliegen und die phantastischsten Muster erzeugen, Moiré genannt. Man spricht deshalb von Moiré-Grafik.

Hier noch ein kleines Programm, daß die Brüche in den Linien für einen anderen Effekt ausnutzt:

```
10 GRAPHICS 24:POKE 710,0
20 FOR Y=0 TO 180 STEP 10
30 COLOR 1:PLOT 0,Y
40 DRAWTO 319,Y+11
50 PLOT 0,191-Y
60 DRAWTO 319,180-Y
70 NEXT Y
80 GOTO 80
```

2.2 Modell mit 16 Gängen

Heim-Computer verfügen üblicherweise nur über zwei Darstellungsarten. In der Grundeinstellung können 40 mal 24 (=960) Zeichen auf dem Bildschirm dargestellt werden, bei den farbenfrohen Modellen in sechzehn Standardfarben. Daneben gibt es dann die (hochauflösende) HIRES-Grafik mit ca. 320 mal 200 Bildpunkten ohne Farbwahl und oft nur mit erheblichem Aufwand zu programmieren.

Um in dieses starre Schema etwas Bewegung zu bringen, erlauben diese Geräte die Definition sogenannter Sprites, Zeichen, die in Form, Farbe und Bewegung vom übrigen Bildschirminhalt getrennt programmiert werden können. Das Programmieren der Sprites ist aber oft auch ein Kapitel für sich.

Statt Sprites bietet ATARI die PM- (Player und Missile) Grafik (s. 4.1.3), eine Routine, die über BASIC nur mit vielen PEEKs, POKEs und Problemen angesprochen werden kann. Bei den Farben aber tut sich der ATARI mit 16 mal 16 Tönen hervor. Und bei der Grafikauflösung fährt er der Konkurrenz mit 16 Gängen davon.

Alle sechzehn Betriebsarten sind zumindest bei den neueren XL-Modellen in BASIC mit GRAPHICS ansprechbar. Obwohl der verantwortliche Chip (ANTIC) der alte geblieben ist, konnte man bei den Vorläufermodellen nur die Betriebsarten 0 bis 11 aufrufen.

Es gibt zwei Grundformen von Betriebsarten. Die eine dient vorrangig der Darstellung von Text. In jede Schreibposition kann ein definiertes Zeichen gePRINTet werden. Die Zeichen werden dem Zeichensatz entnommen. Unter Umständen kann ein spezieller Zeichensatz definiert werden. Die fünf verfügbaren Betriebsarten (0, 1, 2, 12 und 13) unterscheiden sich nur in der Größe der Schreibpositionen und den Farbmöglichkeiten.

Die zweite Grundform dient der Darstellung von Grafik, die aus einzelnen Grafikpunkten zusammengesetzt wird. Die verschiedenen Grafikmodi unterscheiden sich in der Größe der Grafikpunkte (Auflösung) und der Anzahl der gleichzeitig darstellbaren Farben.

Grafik-modus	Kurzbeschreibung	Auflösung Spalten	Auflösung Zeilen	Farbregister* 708	709	710	711	712
0	Standard-Text	40	20/24		*	*		R
1	doppelte Breite	20	20/24	A	a	A	a	H
2	doppelte Größe	20	10/12	A	a	A	a	H
12	bunte Zeichen	40	20/24	A/A	A/A	A	A	H
13	bunte Zeichen	40	10/12	A/A	A/A	A	A	H
4	Zwei-Farben-Grafik	80	40/48	CO.1				H
6		160	80/96	CO.1				H
14		160	160/192	CO.1				H
3	Vier-Farben-Grafik	40	20/24	CO.1	CO.2	CO.3		H
5		80	40/48	CO.1	CO.2	CO.3		H
7		160	80/96	CO.1	CO.2	CO.3		H
15		160	160/192	CO.1	CO.2	CO.3		H
9	Viel-Farben-Grafik	80	192	16 Helligkeitsstufen s. 2.2.9				
10		80	192	9 Farbtöne s. 2.2.10				
11		80	192	16 Standard-Farben s. 2.2.11				
8	HIRES	320	160/192		CO.1	CO.0		R

*Alle Angaben beziehen sich auf das Grafikfenster. In GR.0 und in allen Modi mit Textfenster bestimmen die Register 710 die Schrift- und Hintergrundfarbe und 709 die Schrifthelligkeit. R = Randfarbe. H = Hintergrundfarbe

In den Grafikmodi 0 bis 8 und 12 bis 15 ist am unteren Rand ein Textfenster abgeteilt, das immer aus vier Zeilen in GR.0 besteht. Der Farbton des Textfensters und die Helligkeit der Schrift wird von den Farbregistern des Grafikfensters mitbestimmt. Gibt man dem Grafikfenster den gleichen Farbton wie dem Hintergrund, verschwindet es optisch und man erhält eine einheitliche Bildfläche die aus zwei verschiedenen Darstellungsarten zusammengesetzt ist.

Das Textfenster kann ausgeschaltet werden, indem man der Kennzahl des GRAPHICS den Wert 16 addiert. Das Grafikfenster wird dann entsprechend größer. Also GR.5 schaltet Grafikmodus 5 mit vier Zeilen Textfenster ein. GR.5+16 (oder GR.21) teilt den gesamten Bildschirm in GR.5-Zeilen ein.

Jeder GRAPHICS-Befehl löscht sofort den gesamten Bildschirm. Doch das Löschen des Bildschirms kann unterbunden werden, indem der Kennzahl des Befehls der Wert 32 hinzugerechnet wird. Das ist jedoch nur sinnvoll, wenn die Auflösung nicht verändert wird, weil es sonst zu unkontrollierbaren Wirkungen kommt. Diese Möglichkeit dient nur dem Einblenden des Textfensters:

```
10 GRAPHICS 19
20 COLOR 1
30 PLOT 0,19
40 DRAWTO 39,19
50 PLOT 10,20
60 DRAWTO 29,20
70 FOR P=0 TO 200
80 NEXT P
90 GRAPHICS 35
100 POKE 710,0
110 POKE 752,1
120 ? "          ! TEXT EINBLENDUNG !"
130 FOR P=0 TO 700
140 NEXT P
150 GRAPHICS 51
160 COLOR 2
170 PLOT 0,20
180 DRAWTO 19,20
190 GOTO 190
```

10: GR.3 ohne Textfenster (+17) wird eingeschaltet und mit

20: Farbe 1

30: vom rechten Rand in Zeile 19

40: bis zum linken Rand derselben Zeile ein Balken gezogen.

50: zeichnet in Zeile 20 von Spalte 10

60: bis Spalte 29 einen zweiten Balken. Dieser Balken liegt an einer Stelle, wo das Textfenster hinreicht, wenn es eingeschaltet ist.

70: Eine kleine Pause,

80: damit unser Auge folgen kann.

90: Jetzt wird GR.3+32 eingeschaltet, also GR.3 mit Textfenster aber ohne den Bildschirm zu löschen.

100: setzt die Farbe des Textfensters auf die Farbe des Hintergrundes, damit das Textfenster nicht als farbig hervorgehobener Kasten erscheint. Nach jedem GRAPHICS müssen definierte Farbwerte (und alle anderen POKEs) erneuert werden.

110: macht den Cursor unsichtbar

120: PRINTet einenText aus. Im Textfenster ist der POS.-Befehl nicht wirksam. Um den kurzen Text fein in die Mitte der Zeile zu setzen, beginnt der String mit entsprechend vielen Leerzeichen. Der Text ist von ! zu ! 20 Zeichen lang, es bleiben also 20 Zeichen übrig, die gleichmäßig auf jede Seite verteilt werden sollen, also je zehn. Da der PRINT aber links in Spalte 2 beginnt (wenn wir den linken Rand nicht umPOKEn), muß der String mit acht Leerzeichen beginnen.

130 und 140: machen mal wieder eine Pause.

150: schaltet GR.3+16+32 ein, also ohne Textfenster und ohne Bildschirmlöschen. Das Textfenster und mit ihr die Texteinblendung verschwindet wieder, die ursprüngliche Grafik ist wieder ganz zu sehen.

160: Mit Farbe 2 wird in

170 und 180: ein gelber Farbbalken gePLOTtet.

Um den Unterschied zu sehen, ändern Sie Zeile 90 in GR.3 und Zeile 150 in GR.19 um, also beide Male mit Bildschirmlöschen und sehen Sie sich die Bescherung an.

Das Grafik- und Textfenster (bzw. das Grafikfenster allein) bedecken nicht die gesamte Fläche des Fernsehschirmes. In GRAPHICS 0 und 8 (bzw. 24) kann diese Randfarbe gesondert bestimmt werden. In den übrigen Begriebsarten sind Rand- und Hintergrundfarbe gleich.

2.2.0 GRAPHICS 0

Dieser Textmodus ist der Grundzustand. Er wird beim Einschalten des Computers, nach einem Warmstart (Taste RESET) und in den Grafikmodi ohne Textfenster auch bei Programmabbruch durch ERROR- oder BREAK-Taste und bei Programmende angewählt. GR.0 muß also nur programmiert werden, wenn vorher eine andere Grafikbetriebsart eingeschaltet war oder wenn in GR.0 mit dem COLOR-Befehl gearbeitet werden soll.

Die Eigenschaften von GR.0 gelten auch sinngemäß für die Textfenster in allen Grafikgängen.

Spalten 40 (0 bis 39)

Die Anfangswerte für den rechten und linken Rand können durch POKE 82,n (linker Rand) und POKE 83,n (rechter Rand) geändert werden.

Zeilen 24 (0 bis 23)

oder 20 Zeilen Grafikfenster plus vier Zeilen Textfenster (kann nur durch POKE 703,4 eingestellt werden)

Auflösung 8 Bildpunkte breit
8 Bildschirmzeilen hoch

Farbregister Zeichenhelligkeit (709)
Hintergrund (710)
Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster COLOR und PLOT	Textfenster PRINT
708	0			
709	1	ATASCII	Schrifthelligkeit	
710	2	ATASCII	Farbton	
711	3			
712	4		Randfarbe	

Neben der Möglichkeit mit ? "Strings" auf den Bildschirm zu schreiben, kann auch der COLOR-Befehl verwendet werden. Dann muß aber auf jeden Fall vorher im Programm GR.0 aufgerufen worden sein. Hinter COLOR wird der ATASCII-Code des Zeichens gesetzt, das dann mit PLOT und DRAWTO ausgegeben werden kann. Den vollständigen ATARI-Zeichensatz zeigt Ihnen folgendes Programm (s. auch Anhang 1):

```
10 GRAPHICS 0
20 POKE 752,1
30 FOR C=0 TO 255
40 COLOR C
50 PLOT 19,11
60 POSITION 21,11
70 ? C
80 OPEN #1,4,0,"K:"
90 GET #1,T:CLOSE #1
100 IF T=32 THEN 120
110 GOTO 80
120 NEXT C
```

10: GRAPHICS 0 einschalten.

20: Cursor unsichtbar machen.

30: Schleifenzähler C von 0 bis 255 (der Zeichensatz umfaßt 256 Zeichen)

40: ruft durch COLOR das zum ATASCII-Wert gehörende Zeichen auf und

50: PLOTtet es an die Stelle 19,11 (Mitte des Bildschirms)

60: Der POS.-Befehl stellt den unsichtbaren Cursor auf 21,11 (das ist neben dem gerade ausgegebenen Zeichen) und

70: ?tet den Wert von C, das ist der ATASCII-Code dieses Zeichens.

80 bis 110: unterbrechen die FOR-NEXT-Schleife so lange, bis die Leertaste gedrückt wird. Diese Routine wird später noch erklärt werden. Drücken Sie jetzt einfach die Leertaste und freuen Sie sich, daß dann erst das nächste Zeichen auf dem Bildschirm erscheint.

120: fragt den Schleifenzähler ab.

Und hier noch ein Beispiel, was DRAWTO in GR.0 bewirkt:

```
10 GRAPHICS 0
20 POKE 752,1
30 COLOR 31
40 PLOT 19,0
```

```
50 DRAWTO 30,11
60 DRAWTO 19,22
70 DRAWTO 8,11
80 DRAWTO 19,0
90 GOTO 90
```

10: schaltet GRAPHICS 0 ein und

20: den Cursor aus

30: ruft ATASCII-Zeichen 31 auf. Diesen Wert hat die Taste "Cursor nach rechts" und wird grafisch durch einen Pfeil dargestellt, der nach rechts weist.

40: schreibt diese Zeichen in die Mitte der oberen Bildschirmkante,

50: zieht eine diagonale Linie nach links. Diese Linie besteht natürlich aus dem aufgerufenen Zeichen.

60: zieht die Linie weiter bis zur Mitte des unteren Bildschirmrandes und

70: geht diagonal hoch nach links.

80: vollendet das Karo.

90: verhindert die Beendigung des Programms.

Gehen wir noch einen Schritt weiter:

```
90 COLOR 147
100 PLOT 0,23
110 DRAWTO 39,0
120 ? "ENDE"
130 GOTO 130
```

90: ruft das Zeichen + in inverser (negativer) Darstellung auf und

100: PLOTtet dieses Zeichen in die untere linke Ecke des Bildschirms.

110: zieht eine Linie mit diesem Zeichen nach oben rechts. Bei dieser groben Auflösung sehen Sie die Brüche in der Linie sehr deutlich und können auch beobachten, wie der Rechner die Koordinatenwerte rundet.

120: schreibt ENDE, aber wohin?

130: fängt das Programm auf.

Lassen Sie das Programm laufen, drücken Sie Taste BREAK, um es zu unterbrechen und dann tasten Sie LIST, um eine weitere Änderung vorzunehmen. Doch was geschieht?

ERROR- LIST

Bevor wieder etwas auf dem Bildschirm ausgegeben werden kann, müssen Sie die Mattscheibe löschen: SHIFT und CLEAR und dann: LIST. Ändern Sie Zeile

```
10 GRAPHICS 0:POKE 703,4
```

Dadurch wird der Bildschirm in ein Grafikfenster mit 20 Zeilen und ein Textfenster mit vier Zeilen aufgeteilt.

Wo steht das ENDE jetzt?

Und warum?

?

2.2.1 GRAPHICS 1/17

In diesem Textmodus sind die Zeichen doppelt so breit wie in Gr.0 und haben dieselbe Höhe. Es ist jeweils nur der halbe Zeichensatz verfügbar, im Grundzustand Großbuchstaben, Satzzeichen und Ziffern. Die zweite Hälfte des Zeichensatzes, Kleichbuchstaben und Pseudografik kann durch POKE 756,226 eingeschaltet und mit POKE 756,224 wieder ausgeschaltet werden.

Die Zeichen können mit COLOR n aufgerufen und mit PLOT und DRAWTO ausgegeben werden. Durch den numerischen Wert n wird ein bestimmtes Zeichen und eines von vier Farbregistern angesprochen.

Mit dem Befehl PRINT=6; können die Zeichen auch als String an das Grafikfenster ausgegeben werden. Je nach Art des Zeichens im String (Großbuchstabe, Kleinbuchstabe, normal oder invers) wird ein anderes Farbregister angesprochen. Ausgegeben werden aber immer nur normale Großbuchstaben.

Der Aufruf mit COLOR erlaubt PLOT und DRAWTO. Der Aufruf mit ?=6; gestattet den POSITION-Befehl.

Spalten	20 (0 bis 19)
Zeilen	24 (0 bis 23) oder 20 plus 4 Zeilen Textfenster
Auflösung	16 Bildpunkte breit 8 Bildschirmzeilen hoch
Farbregister	Zeichen (708, 709, 710, 711) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PRINT#6 und PLOT	Textfenster PRINT
708	0	ATASCII	Großb./Ziffern	
709	1	ATASCII	Kleinbuchst.	Schrifthelligkeit
710	2	ATASCII	inv. Großb./Ziff.	Farbton
711	3	ATASCII	inv. Kleinbuchst.	
712	4		Hintergrund	

Die Ermittlung des numerischen Ausdrucks, der hinter COLOR ein bestimmtes Zeichen in einer bestimmten Farbe aufruft ist etwas verwirrend, wenn auch ein System hinter der Sache steckt.

Die normalen ATASCII-Werte 32 bis 95 sprechen die Farbe aus Register 708 (SE.0) an. Teilen wir diesen halben Zeichensatz in Gedanken in zwei Teile. Die obere Hälfte reicht dann vom Leerzeichen (32) bis zum Fragezeichen (63) und die untere Hälfte vom Klammeraffen (64) bis zum Unterstrich (95).

COLOR-Wert	Differenz	Farbregister	SETCOLOR
32 bis 63	ATASCII	708	0
0 bis 31	ATASCII - 32	709	1
160 bis 191	ATASCII + 128	710	2
128 bis 159	ATASCII + 128 - 32	711	3
64 bis 95	ATASCII	708	0
96 bis 127	ATASCII + 32	709	1
192 bis 223	ATASCII + 128	710	2
224 bis 255	ATASCII + 128 + 32	711	3

Der bequemeren Übersicht halber finden Sie im Anhang 5 eine Tabelle, aus der Sie die COLOR-Werte für das jeweilige Zeichen in einer bestimmten Farbe ablesen können. Und mit dem folgenden Prograämmchen können Sie sich den Farb- und Zeichenvorrat anschauen:

```
10 GRAPHICS 1
20 FOR C=0 TO 255
30 COLOR C
40 ? C,
50 PLOT 9,9
60 OPEN #1,4,0,"K:"
70 GET #1,T
80 CLOSE #1
90 IF T=32 THEN 110
100 GOTO 60
110 NEXT C
```

10: schaltet die Grafikbetriebsart 1 ein. Sie können hier aber auch GR.2 einsetzen, dann sind die Zeichen größer und besser zu erkennen.

20: arbeitet alle Werte von 0 bis 255 in der Variablen C ab.

30: ruft Zeichen/Farbe mit dem COLOR-Wert C auf.

40: druckt zur Kontrolle den Wert C ins Textfenster.

50: PLOTtet COLOR C an die Stelle 9,9.

60 bis 100: kontrollieren den Ablauf des Programms über die Leertaste.

110: prüft die Schleifenvariable C.

Wenn Sie sich die erste Hälfte des Zeichensatzes (Großbuchstaben, Satzzeichen, Ziffern) in den Farben der vier Farbregister angesehen haben, können Sie sich noch die zweite Hälfte (Kleinbuchstaben, Pseudografik) ansehen, wenn Sie Zeile 10 ändern:

```
10 GRAPHICS 1:POKE 756,226
```

Es ist in diesem Grafikgang nicht möglich, Zeichen aus der ersten Hälfte des Zeichensatzes und solche aus der zweiten Hälfte gleichzeitig auf die Mattscheibe zu bringen.

Nun versuchen Sie noch die Ausgabe mit PRINT:

```
10 GRAPHICS 1
20 POSITION 6,5
30 ? #6;"QUATSCH"
```

Die POS.itionierung in Zeile 20 stellt den unsichtbaren Cursor auf 6,5. Ab hier wird der String fortlaufend ausgedruckt.

Schreiben Sie jetzt den String in Zeile 30 in inversen Großbuchstaben. RUN. Und jetzt in Kleinbuchstaben. RUN.

Was ist denn mit dem ! los!?

Und jetzt noch in inversen Kleinbuchstaben? (och nö)

2.2.2 GRAPHICS 2/18

Die Angaben für GRAPHICS 1 treffen auch auf GR.2 zu. Der einzige Unterschied besteht in der Auflösung.

Spalten	20 (0 bis 19)
Zeilen	12 (0 bis 11) oder 10 plus 4 Zeilen Textfenster
Auflösung	16 Bildpunkte breit 16 Bildschirmzeilen hoch
Farbregister	Zeichen (708, 709, 710, 711) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PRINT#6 und PLOT	Textfenster PRINT
708	0	ATASCII	Großb./Ziffern	
709	1	ATASCII	Kleinbuchst.	Schrifthelligkeit
710	2	ATASCII	inv. Großb./Ziff.	Farbton
711	3	ATASCII	inv. Kleinbuchst.	
712	4		Hintergrund	

Wenn Sie noch Lust haben, dann sollten Sie einmal dieses hier eintasten (und wenn Sie keine Lust mehr haben, dann legen Sie das Buch beiseite und lesen morgen weiter):

```
10 GRAPHICS 2
40 COLOR 151
50 PLOT 0,0
60 DRAWTO 19,0
70 COLOR 179
80 PLOT 0,1
90 DRAWTO 19,1
100 COLOR 19
110 PLOT 0,2
120 DRAWTO 19,2
130 COLOR 56
140 PLOT 0,3
150 DRAWTO 19,3
```

10: schaltet GR.2 ein

40: wählt das Zeichen 151 (55+128-32). ATASCII 55 ist die Ziffer 7, die Differenz von +96 (128-32) spricht das Farbregister 711 (SE.3, Standardwert Flieder) an.

50: PLOTtet eine lila 7 oben links in die Ecke des Bildschirms und

60: zieht die lila 7, eine neben der anderen, insgesamt 20 Stück, bis in die Ecke oben rechts.

70: ruft Zeichen 51, das ist die Ziffer 3, +128 (=179) auf, das erregt Farbregister 710 (SE.2)

80: bringt die blaue 3 nach 0,1 und Zeile

90: zieht Leine bis zum rechten Rand.

100: ruft Zeichen 51 auf und durch -32 (=19) wird Register 709 (SE.1) abgefragt. Dadurch erscheint in Zeile

110: diesmal eine gelbe 3, die in Zeile

120: wieder bis zum rechten Rand gezogen wird.

130: ruft mit dem Wert 56 die 8 herbei, die aus Register 708 (SE.0) orange gefärbt in Zeile

140: in der vierten Zeile antritt und auch

150: zum linken Rand geDRAWTOt (it's Denglisch!) wird.

Wenn Sie diese schöne Grafik anflimmert, sollten wir noch einen weiteren Schritt wagen. Schalten Sie um, auf die zweite Hälfte des Zeichensatzes:

```
30 POKE 756,226
```

Herzergreifend! Was Ihnen hier um die Augen geschlagen wird, verrate ich nicht. Ich habe nämlich schon gemerkt, daß sie recht faul sind und auf früheren Seiten schon mal das eine oder andere Übungsprogramm geschwänzt haben.

Aber was tun wir nun mit dem Schlamassel auf dem Bildschirm. Ist die zweite Hälfte des Zeichensatzes nur eine theoretische Möglichkeit?

Sie sollten sich jetzt erst einmal im Anhang die Tabelle 2 ansehen. Sie finden in der zweiten Hälfte des Zeichensatzes kein Leerzeichen. Da müssen wir uns einen Trick einfallen lassen. Aber wie!

Am liebsten würde ich Ihnen jetzt vorschlagen, doch selbst über eine Lösung nachzudenken. Aber ich verrat es doch lieber gleich, dann gibt es keinen Ärger.

Man nehme: ein Farbregister, das nicht gebraucht wird. Da wir alle Register im Einsatz haben, müssen wir eines opfern. Soll es 708 treffen. Dieses Register stellen wir auf den gleichen Wert ein, den auch der Hintergrund (712) hat.

Wenn Sie den Wert in 712 nicht kennen, fragen Sie einfach: ?PEEK(712) oder kümmern sich gar nicht drum und schreiben POKE 708,PEEK(712). Falls sich dann im Verlauf des Programmes der Farbwert des Hintergrundes ändern sollte, so ändert sich der Wert in 708 gleich mit.

Da die Herzchen ihre Farbe aus Register 708 beziehen, sind sie zwar immer noch da, aber nicht mehr zu sehen, weil sie die gleiche Farbe wie der Hintergrund haben.

Falls Sie aus irgendeinem unvorstellbaren Grund nicht das Farbregister 708 dem Hintergrund angleichen können (oder wollen), dann müßten Sie nicht nur ein anderes Register opfern sondern auch mit einem COLOR-Wert, der sich auf dieses Register bezieht, alle Leerstellen vollschreiben.

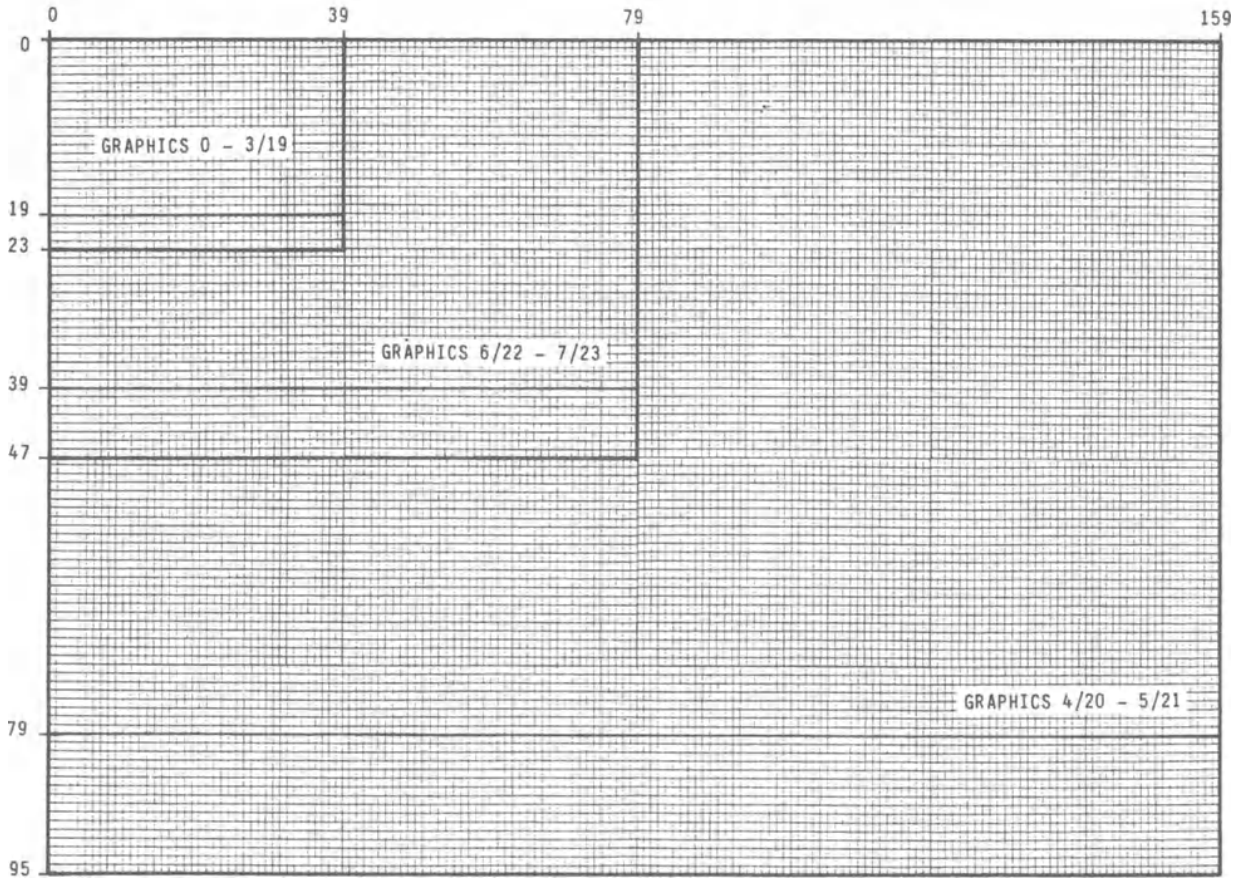
Wenn Sie auch

```
20 POKE 708,0
```

vorziehen, dann können Sie die drei letzten Zeilen getrost löschen:

```
130
140
150
```

Der Mühe Lohn. So ist das beim Programmieren. Mit etwas Ideenreichtum gibt es (fast) immer eine Lösung. Vielleicht ist diese Herausforderung der Grund, warum immer mehr Leuten (warum eigentlich fast nur Männern?) die Finger am Keyboard festwachsen.


0
39
79
159
0
19
23
39
47
79
95
GRAPHICS 0 - 3/19
GRAPHICS 6/22 - 7/23
GRAPHICS 4/20 - 5/21

2.2.3 GRAPHICS 3/19

Diese Betriebsart hat die gröbste Auflösung. Ein Grafikpunkt ist so groß wie der Cursor in GR.0. Für viele Anwendungen reicht diese Auflösung aus. Oft ist eine gröbere Auflösung sogar wirkungsvoller als eine feine. Das muß im Einzelfall der Augenschein zeigen. Besonderer Vorzug dieses Grafikganges ist der trotz Vierfarbigkeit geringe Speicherbedarf von nur 279 Bytes.

Spalten	40 (0 bis 39)
Zeilen	24 (0 bis 23) oder 20 plus 4 Zeilen Textfenster
Auflösung	8 Bildpunkte breit 8 Bildschirmzeilen hoch
Farbregister	Grafikpunkte (708, 709, 710) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0	1	Grafikp./CO.1	
709	1	2	Grafikp./CO.2	Schrifthelligkeit
710	2	3	Grafikp./CO.3	Farbton
711	3			
712	4	0	Hintergrund	

```
100 GRAPHICS 3
110 COLOR 1:PLOT 0,0
120 COLOR 2:PLOT 1,1:DRAWTO 2,1
130 COLOR 3:PLOT 3,2:DRAWTO 6,2
140 ? "DREI FARBBALKEN"
```

2.2.4 GRAPHICS 4/20

Dieser Grafikmodus hat eine doppelt feine Auflösung. Durch Verzicht auf zwei Farbtöne ist der Speicherbedarf für einen Bildschirminhalt mit 537 Bytes auch noch sehr gering.

Spalten	80 (0 bis 79)
Zeilen	48 (0 bis 47) oder 40 plus 4 Zeilen Textfenster
Auflösung	4 Bildpunkte breit 4 Bildschirmzeilen hoch
Farbregister	Grafikpunkte (709) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0	1	Grafikp./CO.1	
709	1	2		Schrifthelligkeit
710	2	3		Farbton
711	3			
712	4	0	Hintergrund	

```
100 GRAPHICS 4
110 COLOR 1:PLOT 0,0:DRAWTO 79,0
120 COLOR 1:PLOT 1,3:DRAWTO 78,3
130 COLOR 3:PLOT 3,6:DRAWTO 76,6
140 ? "DREI FARBBALKEN"
```

2.2.5 GRAPHICS 5/21

Die Angaben für GRAPHICS 4 gelten auch für diese Betriebsart, nur daß vier Farbtöne ausgegeben werden können. Der Bildschirmspeicherbedarf ist dadruch fast doppelt do groß.

Spalten	80 (0 bis 79)
Zeilen	48 (0 bis 47) oder 40 plus 4 Zeilen Textfenster
Auflösung	4 Bildpunkte breit 4 Bildschirmzeilen hoch
Farbregister	Grafikpunkte (708, 709, 710) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0	1	Grafikp./CO.1	
709	1	2	Grafikp./CO.2	Schrifthelligkeit
710	2	3	Grafikp./CO.3	Farbton
711	3			
712	4	0	Hintergrund	

```
100 GRAPHICS 5
110 COLOR 1:PLOT 0,0:DRAWTO 79,0
120 COLOR 2:PLOT 1,3:DRAWTO 78,3
130 COLOR 3:PLOT 3,6:DRAWTO 76,60
140 ? "DREI FARBBALKEN"
```

2.2.6 GRAPHICS 6/22

Dieser Grafikgang mit vierfacher Auflösung hat einen Speicherbedarf von knapp 2 kByte, obwohl nur eine Farbe für Grafikpunkte und eine für den Hintergrund gewählt werden kann.

Spalten	160 (0 bis 159)
Zeilen	96 (0 bis 95) oder 80 plus 4 Zeilen Textfenster
Auflösung	2 Bildpunkte breit 2 Bildschirmzeilen hoch
Farbregister	Grafikpunkte (708) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0	1	Grafikp./CO.1	
709	1	2		Schrifthelligkeit
710	2	3		Farbton
711	3			
712	4	0	Hintergrund	

```
100 GRAPHICS 6
110 COLOR 1:PLOT 0,0:DRAWTO 159,0
120 COLOR 1:PLOT 2,5:DRAWTO 157,5
130 COLOR 1:PLOT 6,10:DRAWTO 153,10
140 ? "DREI FARBBALKEN"
```

2.2.7 GRAPHICS 7/23

Die Angaben von GR.6 entsprechen auch dieser Betriebsart. Es können jedoch vier Farbtöne gleichzeitig dargestellt werden. Der Speicherbedarf für den Bildschirm ist deshalb fast doppelt so groß (3945 Byte)

Spalten	160 (0 bis 59)
Zeilen	96 (0 bis 95) oder 80 plus 4 Zeilen Textfenster
Auflösung	2 Bildpunkte breit 2 Bildschirmzeilen hoch
Farbregister	Grafikpunkte (708, 709, 710) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0	1	Grafikp./CO.1	
709	1	2	Grafikp./CO.2	Schrifthelligkeit
710	2	3	Grafikp./CO.3	Farbton
711	3			
712	4	0	Hintergrund	

```
100 GRAPHICS 7
110 COLOR 1:PLOT 0,0:DRAWTO 159,0
120 COLOR 2:PLOT 2,5:DRAWTO 157,5
130 COLOR 3:PLOT 6,10:DRAWTO 153,10
140 ? "DREI FARBBALKEN"
```

2.2.8 GRAPHICS 8/24

Bei dieser feinsten Auflösung (HIgh RESoslution) kann jeder der 61.440 Bildpunkte (Pixel) einzeln angesprochen werden. Das erfordert einen Speicherplatz für den Bildschirm von fast 8 kByte (7900 Byte). Deshalb kann nur eine Farbe für das Grafikfenster und die Helligkeit der Grafikpunkte definiert werden.

Diese Betriebsart ist von der Struktur her mit GRAPHICS 0 gleich. So ist es z.B. denkbar, in GR.8 Zeichen Punkt für Punkt darzustellen und das könnten auch Zeichen sein, die dem Zeichensatz entsprechen. Es wäre allerdings eine sehr mühsame Arbeit, einen ganzen Text, Buchstabe um Buchstabe, und jeden einzelnen Buchstaben Pixel für Pixel zu programmieren.

Spalten	320 (0 bis 319)
Zeilen	192 (0 bis 191) oder 160 plus 4 Zeilen Textfenster
Auflösung	jeder einzelne Bildpunkt kann für sich angesprochen werden
Farbregister	Grafikpunkthelligkeit (709) Hintergrund (710) Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0			
709	1	1	Grafikp.hell	Schrifthelligkeit
710	2	0	Hintergrund	Farbton
711	3			
712	4		Randfarbe	

```
100 GRAPHICS 24
110 COLOR 1
120 FOR X=0 TO 17
130 FOR B=0 TO X
140 PLOT X^2+B,0
150 DRAWTO X^2+B,191
160 NEXT B
170 NEXT X
175 ? "Was ist jetzt los?"
180 FOR CH=6 TO 254 STEP 8
190 POKE 710,CH
200 POKE 712,CH-6
210 FOR CG=0 TO 14 STEP 2
220 POKE 709,CG
230 FOR PAUSE=0 TO 300
240 NEXT PAUSE
250 NEXT CG
260 NEXT CH
270 GOTO 180
```

100: GR.8+16 (ohne Textfenster)

110: Warum COLOR? Es steht zwar nur eine Farbe (bzw. abweichende Helligkeit) für Grafikpunkte zur Verfügung, aber wir können auch COLOR 0 (Hintergrund) aufrufen und damit gePLOTtetes ausradieren.

120 bis 170: erzeugen ein Bild aus immer breiter werdenden Streifen (beachten Sie den räumlichen Effekt!)

175: Ja, was ist jetzt los! Wo soll denn der arme Rechner diesen Text hindrucken. Wir haben ihm kein Textfenster gelassen, also schaltet er GR.0 ein, um diesen Auftrag auszuführen. Löschen Sie diese Zeile oder richten Sie ein Textfenster ein.

180 bis 260: wechselt die Farbwerte für den Hintergrund, den Rand und die Helligkeit der Grafikpunkte. Wenn die Grafikpunkte die gleiche Helligkeit haben wie der Hintergrund, dann sehen Sie gar nichts, denn der Farbton von beiden ist gleich.

270: Rücksprung nach 180.

2.2.9 GRAPHICS 9

In dieser Betriebsart kann nur ein Farbton ausgewählt werden, aber mit COLOR 0 bis 15 können alle sechzehn Helligkeitsstufen auf dem Bildschirm dargestellt werden. Dies ist der einzige Grafikmodus, bei dem die ungeraden Helligkeitswerte zur Wirkung kommen.

Die Grafikpunkte haben eine etwas unhandliche Form, aber nur so läßt sich die relativ feine Auflösung so farbfreudig realisieren, ohne den Speicher zum Platzen zu bringen. Immerhin werden durch GR.9 7900 Bytes belegt.

Spalten	80 (0 bis 79)
Zeilen	192 (0 bis 191) Textfenster nicht möglich
Auflösung	4 Bildpunkte breit 1 Bildschirmzeile hoch
Farbregister	Farbton der Grafikpunkte, Rand und Hintergrund (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO
712	4	0	Grafikpunkte/Hintergrund
		1	Grafikpunkt Helligkeit 1
		2	Grafikpunkt Helligkeit 2
		3-14	Helligkeiten 3 bis 14
		15	Grafikpunkt Helligkeit 15

Mit dem folgenden Demonstrationsprogramm können Sie sich den kompletten Farbreichtum des ATARI vorführen lassen. Das Programm hat aber auch noch einen praktischen Nutzen: Sie können Ihren Fernseher oder Monitor auf die Farbsignale des ATARI abstimmen. Der Bildschirm ist in sechzehn Farb-

streifen mit steigenden Helligkeiten aufgeteilt:

```
100 GRAPHICS 9
110 POKE 712,0
120 FOR H=0 TO 15
130 COLOR H
140 FOR B=0 TO 4
150 PLOT H*5+B,0
160 DRAWTO H*5+B,191
170 NEXT B
180 NEXT H
190 FOR C=0 TO 15
200 POKE 712,C*16
210 FOR PAUSE=0 TO 300
220 NEXT PAUSE
230 NEXT C
240 GOTO 190
```

120: wechselt den Helligkeitswert H von 0 bis 15, der

130: mit COLOR aufgerufen wird

140: sorgt dafür, daß Farbbalken gePLOTtet werden, die fünf Grafikpunkte breit sind. Das geschieht in 150 und 160.

190: wechselt den Farbwert C von 0 bis 15.

Das nächste Programm soll Ihnen zeigen, was man mit GR.9 anstellen kann (wenn man will):

```
10 REM VORSPANN
20 GRAPHICS 9:SETCOLOR 4,1,0
40 C=C+1:IF C=16 THEN 230
50 COLOR C:A=A+1
60 FOR L=191 TO 0 STEP -19
70 PLOT 24+A,95:DRAWTO 0,L:NEXT L
100 FOR O=0 TO 79 STEP 8
110 PLOT 24+A,95:DRAWTO O,0:NEXT O
140 FOR R=1 TO 191 STEP 19
150 PLOT 24+A,95:DRAWTO 79,R:NEXT R
180 FOR U=79 TO 0 STEP -8
190 PLOT 24+A,95:DRAWTO U,191:NEXT U
220 GOTO 40
230 GOTO 230
```

Von einem Punkt in mittlerer Höhe des Bildschirms (24+A,95) werden strahlenförmig Linien zum L(inken), O(beren), R(echten) und U(nteren) Bildschirmrand gezogen. Wenn ein Strahlenbündel fertiggePLOTtet ist, wird mit COLOR C die nächste Helligkeitsstufe gewählt und das Zentrum der Strahlen um einen Grafikpunkt nach rechts verlegt. (Die Variable A ist überflüssig PLOT 24+C,95 bringt das selbe Ergebnis!)

2.2.10 GRAPHICS 10

Dieser Grafikgang ist GRAPHICS 9 verwandt. Die Auflösung ist identisch, es können jedoch neun verschiedene Farbtöne gleichzeitig dargestellt werden, deren Farbwerte alle beliebig definierbar sind (aber nur gerade Werte).

Mit dem SE.-Befehl können jedoch nur fünf Farbwerte festgesetzt werden. Die übrigen vier Farbtöne können nur durch direkten Speicherzugriff gePOKEt werden. Die zusätzlichen Register 704 bis 707 werden nur in diesem Grafikmodus angesprochen. Sie stehen sonst für die Player-Missile-Grafik (s. 4.1.3) zur Verfügung.

Die Register 704 bis 707 haben den Anfangswert 0 (Schwarz)

Spalten	80 (0 bis 79)
Zeilen	192 (0 bis 191) Textfenster nicht möglich
Auflösung	4 Bildpunkte breit 1 Bildschirmzeile hoch
Farbregister	Grafikpunkte (705 bis 712) Hintergrund und Rand (704)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO
704		0	Grafikpunkt/Hintergrund
705		1	Grafikpunkt
706		2	Grafikpunkt
707		3	Grafikpunkt
708	0	4	Grafikpunkt
709	1	5	Grafikpunkt
710	2	6	Grafikpunkt
711	3	7	Grafikpunkt
712	4	8	Grafikpunkt

```
100 REM GR10DEMO.BIR
110 GRAPHICS 10
120 FOR I=0 TO 8
130 POKE 704+I,12*I
140 NEXT I
150 FOR C=0 TO 8
160 COLOR C
170 FOR Y=0 TO 19
180 PLOT 0,C*20+Y
190 DRAWTO 79,C*20+Y
200 NEXT Y
210 NEXT C
220 COLOR 0
230 PLOT 0,0
240 DRAWTO 79,179
250 FOR J=0 TO 100
260 FOR I=0 TO 8
270 TRAP 100
280 POKE 704+I,PEEK(704+I)+2
300 NEXT I
310 FOR PAUSE=0 TO 300
320 NEXT PAUSE
330 NEXT J
```

100: REM(ember), eine Denkzettelzeile, die vom Rechner nicht bearbeitet wird.

120 bis 140: POKEt in die Farbregister 704+0 bis 704+8 (=712) die Farbwerte 12*0 bis 12*8.

150: wechselt den COLOR-Wert C von 0 bis 8

170 bis 190: zeichnen einen waagerechten Farbbalken, der zwanzig (Y=0 bis 19) Bildschirmzeilen hoch ist. Der Wert C*20 sorgt dafür, daß jede 20ste Zeile ein neuer Farbbalken beginnt.

220 bis 240: ziehen eine Diagonale in der Farbe 0 (Hintergrund) durch das Grafikfenster.

250: Für 101 Durchläufe

260: wird mit der Variablen I die Farbe in den Farbregistern gewechselt.

270: Dabei wird irgendwann in Zeile 280 ein unzulässiger Farbwert gePOKEt. TRAP sorgt vor: Wenn ein Fehler auftritt, wird nicht etwa das Programm abgebrochen, das wäre ja auch schade, sondern nach Zeile 100 verzweigt. Die TRAP-Falle muß natürlich vor einer möglichen Fehlerquelle lauern.

2.2.11 GRAPHICS 11

Auch diese Betriebsart ist GRAPHICS 9 und 10 verwandt. Die Auflösung ist die gleiche, es können jedoch sechzehn verschiedene Farbtöne gleichzeitig auf der Mattscheibe dargestellt werden. Allerdings lassen sich diese sechzehn Farben nicht beliebig bestimmen. Die sechzehn Farben sind auf die sechzehn Grund-Farbwerte (0 bis 240 STEP 16) eingestellt. Nur die Helligkeit kann programmiert werden, gilt dann aber für alle sechzehn Farbtöne gleichzeitig.

Spalten	80 (0 bis 79)
Zeilen	192 (0 bis 191) Textfenster nicht möglich
Auflösung	4 Bildpunkte breit 1 Bildschirmzeile hoch
Farbregister	Helligkeit für alle Grafikpunkte Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO
712	4	0	Grafikp.hell/Hintergr. Farbe 0
		1	Grafikpunkt Farbe 16
		2	Grafikpunkt Farbe 32
		3-14	Grafikpunkte Farben 48 bis 224
		15	Grafikpunkt Farbe 240

Mit dem folgenden Demonstrationsprogramm können Sie sich die Farbmöglichkeiten dieses Grafikmodus' ansehen. Es wird ein Testbild aufgebaut, das alle sechzehn Farbtöne zeigt. Die Diagonale in der Hintergrundfarbe (COLOR 0) zeigt die Größe des Grafikfensters. Der erste Farbbalken hat die Farbe des Hintergrundes und geht deshalb optisch im gleichfarbigen Bildschirmrand unter.

```
100 REM GR11DEMO.BIR
110 GRAPHICS 11
120 FOR C=0 TO 15
130 COLOR C
140 FOR F=0 TO 4
150 PLOT C*5+F,0
160 DRAWTO C*5+F,191
170 NEXT F
180 NEXT C
190 COLOR 0
200 PLOT 0,0
210 DRAWTO 79,191
220 FOR H=2 TO 12 STEP 2
230 POKE 712,H
240 FOR PAUSE=0 TO 300
250 NEXT PAUSE
260 NEXT H
270 FOR H=14 TO 0 STEP -2
280 POKE 712,H
290 FOR PAUSE=0 TO 300
300 NEXT PAUSE
310 NEXT H
320 GOTO 220
```

120: Diese Schleife geht mit C die Wert 0 bis 15 durch, die

130: als COLOR-Wert aufruft.

140: Fünf (0 bis 4) Grafikpunkte nebeneinander werden

150: Linien vom oberen

160: bis zum unteren Bildschirmrand gezogen.

170 und 180: Und zwar nebeneinander in den verschiedenen COLOR-Werten 0 bis 15

190 bis 210: ziehen eine Diagonale in der Hintergrundfarbe (COLOR 0) durch das Grafikfenster

220: Hier werden die Helligkeitswerte aufsteigend gewählt,

230: gePOKEt

240 und 250: und nach einer Augenpause

260: gewechselt.

270 bis 310: lassen die Farben von dunkel nach hell laufen.

320: Und hier beginnt der ganze Spaß von vorn.

2.2.12 GRAPHICS 12/28

Dieser (und der folgende) Grafikmodus ist für die Darstellung gestalteter Zeichen vorgesehen und mit den Betriebsarten GR.1 und 2 verwandt. Die technisch richtige Bezeichnung als Textbetriebsart ist jedoch etwas irreführend. Es können zwar die 256 Zeichen des Zeichensatzes aufgerufen werden, aber was auf der Mattscheibe ankommt, ist nur mit Mühe als Zeichen zu erkennen.

Die Zeichen setzen sich aus Farbpunkten dreier verschiedner Farbtöne zusammen und zwar die erste Hälfte des Zeichensatzes, also die Zeichen in normaler Darstellung, enthält Farbanteile aus den Registern 708, 709 und 710, während die Zeichen aus der zweiten Hälfte des Zeichensatzes, die inversen, von den Registern 708, 709 und 711 gefärbt werden.

Spalten	40 (0 bis 39)
Zeilen	24 (0 bis 23) oder 20 plus 4 Zeilen Textfenster
Auflösung	8 Bildpunkte breit 8 Bildschirmzeilen hoch
Farbregister	normale Zeichen (708, 709, 710) inverse Zeichen (708, 709, 711) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PRINT#6 und PLOT	Textfenster PRINT
708	0	ATASCII	alle Zeichen	
709	1	ATASCII	alle Zeichen	Schrifthelligkeit
710	2	ATASCII	normale Zeichen	Farbton
711	3	ATASCII	inverse Zeichen	
712	4		Hintergrund	

Die Anwendung dieses Grafikganges ist eigentlich nur interessant, wenn dafür ein spezieller Zeichensatz definiert wird, der die farblichen Möglichkeiten in ihrer Wirkung einbezieht. Ist ein solcher anwendungsorientierter Zeichensatz erst einmal erarbeitet, lassen sich damit allerdings sehr wirkungsvolle Grafikeffekte leicht und schnell-laufend programmieren.

Bescheiden wir uns also vorläufig damit anzuschauen, was diese beiden Grafikgänge aus den Daten des Standard-Zeichensatzes machen:

```
10 GRAPHICS 12+16
20 FOR Y=0 TO 23
30 FOR X=0 TO 39
40 OPEN #1,4,0,"K:"
50 GET #1,T
60 CLOSE #1
70 COLOR T
80 PLOT X,Y
90 NEXT X
100 NEXT Y
110 FOR C=0 TO 20
120 POKE 712,PEEK(712)+10
130 POKE 708,PEEK(708)+8
140 POKE 709,PEEK(709)+2
150 POKE 710,PEEK(710)+4
160 POKE 711,PEEK(711)+6
170 FOR P=0 TO 400
180 NEXT P
190 NEXT C
200 END
```

20: Zeile um Zeile (0 bis 23)

30: Spalte neben Spalte (0 bis 39)

40 bis 60: wird die Tastatur abgefragt, welche Taste gedrückt wurde und dann mit

70: COLOR das zugehörige Zeichen

80: auf den Bildschirm gePLOTtet.

110 bis 190: Wenn auf diese Weise der Bildschirm vollgeschrieben worden ist, werden die Farbwerte in den Farbregistern gewechselt. Und zwar wird in ein Register hineingeschaut (PEEK(712)) ein Wert hinzugezählt (+12) und in das gleiche Register wieder hineingeschrieben (POKE 712,PEEK (712)+10).

2.2.13 GRAPHICS 13/29

Die Angaben zu GRAPHICS 12 treffen auch auf diesen Grafikmodus zu. Der einzige Unterschied besteht in der Auflösung.

Spalten	40 (0 bis 39)
Zeilen	12 (0 bis 11) oder 10 plus 4 Zeilen Textfenster
Auflösung	8 Bildpunkte breit 16 Bildschirmzeilen hoch
Farbregister	normale Zeichen (708, 709, 710) inverse Zeichen (708, 709, 710)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PRINT#6 und PLOT	Textfenster PRINT
708	0	ATASCII	alle Zeichen	
709	1	ATASCII	alle Zeichen	Schrifthelligkeit
710	2	ATASCII	normale Zeichen	Farbton
711	3	ATASCII	inverse Zeichen	
712	4		Hintergrund	

Das folgende Demonstrationsprogramm druckt den gesammten Zeichensatz auf den Bildschirm. Der Bediener kann die Ausgabe kontrollieren. Erst nachdem eine beliebige Taste der Tastatur gedrückt wurde, wird das nächste Zeichen auf dem Bildschirm dargestellt. Eine Beziehung zwischen der gedrückten Taste und dem dargestellten Zeichen gibt es bei diesem Programm (im Gegensatz zum vorherigen) nicht.

Im Textfenster wird gleichzeitig der ATASCII-Wert und das Zeichen selbst in GR.0-Darstellung gezeigt.

Wenn alle 400 Schreibpositionen vollgeschrieben sind, auf dem Bildschirm ist dann der komplette Zeichensatz etwa einundeinhalbmal zu sehen, werden die Farbwerte in den Farbregistern laufend gewechselt.

```
10 GRAPHICS 13
20 FOR Y=0 TO 9
30 FOR X=0 TO 39
40 C=C+1
50 IF C=125 THEN C=126
60 IF C>255 THEN C=C-256
70 COLOR C
80 PLOT X,Y
90 ? C,CHR$(C)
100 OPEN #1,4,0,"K:"
110 GET #1,DUMMY
120 CLOSE #1
130 NEXT X
140 NEXT Y
150 FOR C=0 TO 14
160 FOR H=0 TO 14
170 FOR P=0 TO 4
180 POKE 708+P,H+C*16+4*P
190 FOR PAUSE=0 TO 300
200 NEXT PAUSE
210 NEXT P
220 NEXT H
230 NEXT C
240 GOTO 150
```

20 und 30: gehen die Koordinaten aller Schreibpositionen nacheinander durch.

40: zählt in die Variable C

50: Der Wert von C wird als ATASCII-Wert verwendet werden und in Zeile 90 wird dieses Zeichen gePRINTet. ATASCII 125 bezieht sich auf die CLEAR-Taste. Wird dieses Zeichen an den Bildschirm ausgegeben (PRINT) so wird der Bildschirm gelöscht. Um das zu vermeiden, wird in dieser Zeile der Wert 125 übersprungen.

60: Da der Zeichensatz nur 256 Zeichen umfaßt, werden größere Werte von C hier entsprechend reduziert. D.h. beim 256sten Durchlauf wird wieder das erste Zeichen aus dem Zeichensatz (ATASCII 0 = Herz) ausgegeben.

90: CHR$(n) verwandelt n in das ATASCII-Zeichen n

100 bis 120: Erst wenn irgendeine Taste gedrückt wurde, läuft das Programm weiter.

150 bis 230: wechseln die Farbwerte in den Registern 708 bis 708+4 (=712).

2.2.14 GRAPHICS 14/30

Diese Auflösung ist fast so fein wie HIRES, bietet aber mehr Farbauswahl.

Spalten	160 (0 bis 159)
Zeilen	192 (0 bis 191) oder 160 plus 4 Zeilen Textfenster
Auflösung	2 Bildpunkte breit 1 Bildschirmzeile hoch
Farbregister	Grafikpunkte (708) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0	1	Grafikp./CO.1	
709	1	2		Schrifthelligkeit
710	2	3		Farbton
711	3			
712	4	0	Hintergrund	

```
80 G=G+1
90 IF G=2 THEN G=0
100 GRAPHICS 14+G+16
110 FOR C=1 TO 3
120 COLOR C
130 PLOT 0+4*C,0
140 DRAWTO 147+4*C,191
150 PLOT 147+4*C,0
160 DRAWTO 0+4*C,191
170 NEXT C
180 FOR P=0 TO 800
190 NEXT P
200 GOTO 80
```

(Erläuterungen auf der Nebenseite)

2.2.15 GRAPHICS 15/31

Der Unterschied zu GR.14 besteht nur darin, daß vier Farbtöne definiert werden können.

Spalten	160 (0 bis 159)
Zeilen	192 (0 bis 191) oder 160 plus 4 Zeilen Textfenster
Auflösung	2 Bildpunkte breit 1 Bildschirmzeile hoch
Farbregister	Grafikpunkte (708, 709, 710) Hintergrund und Rand (712)

Register POKE	SETCOLOR SE.	COLOR CO.	Grafikfenster PLOT und DRAWTO	Textfenster PRINT
708	0	1	Grafikp./CO.1	
709	1	2	Grafikp./CO.2	Schrifthelligkeit
710	2	3	Grafikp./CO.3	Farbton
711	3			
712	4	0	Hintergrund	

Das Demonstrationsprogramm auf der linken Seite zeigt die Wirkung von GR.14 und 15. Die Variable G kann die Werte 0 und 1 annehmen. Dadurch wird in Zeile 100 GR.14 oder GR.14+1 (beide ohne Textfenster) eingeschaltet. Mit Farbe 1 bis 3 werden dann diagonale Linien gezogen. Nach einer Pause wird der andere Grafikmodus eingeschaltet und werden erneut die Diagonalen gezogen. Wenn in Zeile 120 zu große COLOR-Werte auftreten, so führt das nicht zu einem Fehler, sondern der zu große Wert wird dividiert und der Rest bestimmt den Farbwert. In GR.14 wirkt COLOR 2 wie COLOR 0, COLOR 3 wie COLOR 1. Das gilt übrigens grundsätzlich für den COLOR-Befehl. Probieren Sie folgendes Porgramm, in dem COLOR-Werte von 1 bis 40 aufgerufen werden!:

```
5 GRAPHICS 3:FOR C=1 TO 40:COLOR C:PLOT C-1,0:NEXT C
```

3 Wie die Bilder laufen lernen

Gewöhnliche Computer-Grafik bezieht ihren Reiz aus der Ästhetik mathematischer Kurven, die als optischer Ausdruck reiner Zahlenstrukturen ein Wohlgefallen vermitteln. Diese Lust an der Reinheit der Form macht aber Computer-Grafik noch nicht zur Kunst.

Eine neue Dimension bietet der Rechner als kreatives Medium mit der fortlaufenden Veränderung, mit sich prozessual verändernden Bildern, eine Zwischenform von Grafik und Film, die aber zusätzlich die Möglichkeit bietet, daß der Betrachter über die Tastatur aktiv in das ablaufende ästhetische Geschehen eingreift.

Selbst mit recht einfachen Programmstrukturen lassen sich ganz ansehnliche Ereignisse auf den Flimmerkasten zaubern. Die farblich-grafischen Möglichkeiten des ATARI bieten immer neue Anregungen. Jedermann kann sich mit geringem Aufwand so richtig gestalterisch austoben.

Um mich in den folgenden Kapiteln nicht dauernd zu wiederholen, möchte ich es hier und jetzt, ein für alle Mal sagen: Die Bei-Spiel-Programme sollen nur eine Anregung sein. Sie lernen am angenehmsten und schnellsten, wenn Sie herumexperimentieren. Ändern Sie die Werte, Variablen, 'Farb-POKEs und Grafikmodi nach Herzenslust, und sehen Sie sich an, was dabei herauskommt!

Der Computer ist ein Spielzeug. Er dient dem Spieltrieb von Wissenschaftlern, Technikern und Geschäftsleuten. Warum sollten sie nicht auch Ihren Spaß damit haben?

3.1 Druck-Grafik

Die einfachste Methode, ein sich laufend veränderndes Bild zu erzeugen, besteht darin, Pseudografik-Zeichen wie einen fortlaufenden Text auf den Bildschirm zu ?en.

Wir erzeugen per Zufall (ja und wie geht das?) ein Pseudografik-Zeichen, PRINTen es, erzeugen das nächste usf. Da der Bildschirminhalt nach oben gescrollt wird (Denglisch: to scoll, scrollte, gescrollt: rollen) wenn er voll ist und unten endlos wechselnde Zeichen gePRINTet werden, ändert sich das Bild buchstäblich laufend. Tippen Sie mal:

```
90 WAHL=INT(RND(0)*4)
100 INVERS=RND(0)
110 IF WAHL=0 THEN ZEICHEN=9
120 IF WAHL=1 THEN ZEICHEN=11
130 IF WAHL=2 THEN ZEICHEN=12
140 IF WAHL=3 THEN ZEICHEN=15
150 IF INVERS>0.5 THEN ZEIHEN=ZEICHEN+128
160 PRINT CHR$(ZEICHEN);
170 GOTO 40
```

90: Hierwird der Zufall bemüht. Der Befehl dafür heißt RND (RaNDom). Hinter RND muß eine sog. Dummywert stehen; der Wert ist beliebig. Schreiben wir also RND(0). Dieser Befehl erzeugt einen zufälligen Wert zwischen 0 und 1, wobei der Wert 1 nie erreicht wird. Multipliziert man diese Zufallswerte mit irgendeiner Zahl, nehmen wir 4, so erhält man Wert von 0 bis 3,99 Periode 9. Diese Dezimalbrüche wandeln wir in ganze (integer) Zahlen um, indem die Nachkommastellen (Sie denken ja daran, daß das Komma ein Punkt ist) unter den Teppich gekehrt werden. Das besorgt der Befehl INT(eger). Das BASIC-Kommando INT(RND(0)) ergibt also in zufälliger Folge Werte von 0 bis 3. Diese Zeile notiert den gefundenen Wert in der Variablen WAHL.

100: In dieser Zeile spielen wir wieder mit dem Zufall. Die Variable INVERS erhält irgendeinen Wert größergleich 0 und kleiner 1.

110 bis 140: Hier wird der Zufallswert in der Variablen WAHL genutzt, um der Variablen ZEICHEN Werte zuzuordnen.

150: bemüht die Variable INVERS, die in Zeile 100 mit RND(0) gefüllt wurde. Wenn die Variable INVERS einen größeren Wert als 0,5 (=US: 0.5) abbekommen haben sollte, dann soll der Wert von ZEICHEN um 128 erhöht werden. Sie erinnern sich, daß die ATASCII-Zeichen negativ daherkommen, wenn ihr Wert größer als 127 ist? Aha.

160: Endlich ist es so weit. Ein ZEICHEN zufälliger WAHL wird INVERS oder nicht gePRINTet. Allerdings ist da noch ein kleines, aber wirklich nur ein kleines Problem. Wir haben gar keine (Pseudografik-) Zeichen, sondern nur einen schäbigen Variablenwert (9, 11, 12 oder 15). CHR$(n) wandelt den numerischen Wert n in das ATASCII-Zeichen n um. Dazu gibts nebenbei gesagt auch einen Umkehrbefehl: ASC("A") liefert den ATASCII-Wert von A (=65). Beachten Sie in dieser Zeile auch das ; am Ende. Unscheinbar wie es ist, verhindert es doch, daß der Cursor davoneilt. So wird schön brav ein Zeichen hinter das andere gedruckt, bis die Zeile voll ist. Dann kommt die nächste ran. Apropos ran, geben Sie doch endlich RUN!

O.K., berauschend finde ich das auch nicht, aber das ist ja auch erst der Anfang:

```
10 POKE 710,0:POKE 709,0
20 POKE 82,0
30 POKE 752,1
40 ZAEHL=ZAEHL+1
50 IF ZAEHL=240 THEN ZAEHL=0:FARB=FARB+1
60 IF FARB=16 THEN FARB=0
70 POKE 710,FARB*16+6
80 POKE 709,FARB
```

10: Neue Farben.

20: Randfrage. Links auf 0

30: schickt den flatterigen Cursor zum Teufel.

40: Jetzt nehmen wir einen ZAEHLknecht. Bei jedem Durchlauf zählt er 1 höher, aber erst wenn in Zeile

50: ZAEHL auf 240 angeschwollen ist, hat das Folgen. ZAEHL wird auf 0 degradiert, aber jetzt kann der Rest der Zeile abgearbeitet werden und das heißt für FARB 1 rauf.

(Der Wert 240 wurde gewählt, weil er sechs Zeilen á 40 Zeichen entspricht. Ich erwähne das nur, damit Sie sehen, daß ich mir auch Gedanken mache.)

60: FARB werden aber nur Werte bis 16 erlaubt.

70: Jetzt kommt FARBe ins Spiel. In Register 710 wird ein Farbwert gePOKEt. Sie erinnern sich, daß der ATARI sechzehn Farbtöne bereithält, die durch Zahlenwerte dargestellt werden, die durch 16 teilbar sind. Zu diesen Werten wird die Helligkeit als gerade Zahl addiert. Wir zählen zu FARB*16 die Helligkeit 6 hinzu, einen gemäßigten Wert der Mitte.

80: POKEt den Wert FARB in das Register 709 (Schrifthelligkeit). Da hier nur die Helligkeit definiert werden kann, ist es gleichgültig, welcher Farbwert (Vielfaches von 16) hier steht.

Bei jedem 240sten Programmdurchlauf, also alle sechs Bildschirmzeilen, wird (falls Sie das Programm wirklich eingetippt und ein RUN gegeben haben) die Farbe des Hintergrundes gewechselt. Die Helligkeit der Zeichen wechselt nur jede zwölfte Zeile. Warum?

Wenn übrigens in Zeile 80 ein Helligkeitswert erreicht wird, der mit der Helligkeit der Hintergrundfarbe identisch ist, dann sollten Sie sich nicht wundern, wenn Sie nichts sehen, jedenfalls keine Zeichen.

Da hier kein Quiz veranstaltet wird (warum eigentlich nicht?), hier die Auflösung der obigen Frage. Wenn Sie bis hierher gelesen haben, denken Sie ja doch nicht darüber nach: Mit Ausnahme von GR.9 sind nur gerade Werte für die Helligkeit gefragt. Ungerade Werte nimmt der Computer zwar nicht krumm, läßt sie aber links liegen. FARB muß also gerade sein, um in Zeile 80 etwas auszurichten.

Der Hobbytip: Setzen Sie in 110 bis 140 andere Werte ein, stellen Sie mehr Zeichen zur Auswahl (Multiplikator in Zeile 90 zu erhöhen vergessen), lassen Sie die Farben tanzen.

Und so umgehen Sie CHR$(n) in Zeile 160: Schreiben Sie in 110 bis 140 statt der numerischen Variablen ZEICHEN die String-Variable ZEICHEN$. Natürlich muß die irgendwo vorher, am besten schon in Zeile 10 DIMensioniert worden sein. Sie können dann das Zeichen als Stirng eingeben ("z"). Die Pseudografik-Zeichen erreichen Sie über die Tastatur, indem Sie die Taste CONTROL und eine Buchstabentaste gleichzeitig drücken. In Zeile 160 muß es dann heißen PRINT ZEICHEN$;.

3.2 Zufallslösung

Ehrlich gesagt, besonders beeindruckend war die endlose Pseudografik im letzten Abschnitt nicht, oder? Es wird halt immer nur am unteren Rand etwas angefügt und oben verschwindet Zeile um Zeile. Es wäre schon wirkungsvoller, ein stehendes Bild zu haben, das sich ständig verändert.

Aber nichts leichter als das! Statt fortlaufend zu PRINTen, können wir das zufällig ausgewählte Zeichen ja auch an eine zufällig auserwählte Stelle des Bildschirms POS.itionieren und PRINTen.

Hausaufgabe: Ergänzen und verändern Sie das vorherige Programmbeispiel entsprechend.

Wieso lesen Sie denn jetzt hier weiter. Hatte ich Ihnen nicht gesagt, Sie sollen das Programm umschreiben?

Na also! Ich setze mal voraus, daß Sie Ihr umgeschriebenes Programm tatsächlich zum Laufen gebracht haben. Dann wissen Sie jetzt auch, daß ich Recht habe, wenn ich behaupte, daß das nicht besonders wirkungsvoll aussieht. Es tut sich zu wenig.

Deshalb lassen Sie uns nun endlich in die farbige Fülle des ATARI greifen:

```
10 GRAPHICS 3+16
60 X=INT(RND(0)*40)
70 Y=INT(RND(0)*24)
80 C=INT(RND(0)*3)
90 COLOR C+1
100 PLOT X,Y
110 GOTO 60
```

10: schaltet GRAPHICS 3 ein und das Textfester aus. Sie wissen natürlich längst, daß Sie den Wert auch selbst addieren dürfen (3+16=32, nur für den Fall, daß Sie keinen Taschenrechner zur Hand haben)

60: Hier wird ein zufälliger Wert zwischen 0 und 39 ausgewählt (das kennen Sie ja schon) und das entspricht rein zufällig den möglichen Spaltenpositionen (0 bis 39)

70: Das gleiche geschieht hier mit den Zeilenpositionen.

80: Schließlich lassen wir Freund Computer noch einen Wert zwischen 0 und 2 auswürfeln, der in Zeile

90: für Farbe sorgt. Da für Grafikpunkte die Register 708, 709 und 710 verantwortlich sind und die mit den Werten COLOR 1, 2 und 3 aufgerufen werden, muß die Variable C um den Wert 1 erhöht werden. Diese Addition hätte natürlich auch schon in Zeile 80 untergebracht werden können, so daß C Werte von 1 bis 3 erhält.

100: PLOTtet einen Grafikpunkt in der Farbe C+1 an die vom RND-Schicksal vorbestimmte Stelle X,Y.

110: Ein Grafikpunkt wäre geschafft. Jetzt schnell zurück nach 60 und den nächsten Punkt zusammengebraut.

Wenn Sie das Programm RUNnen, regnet es Farbe, aber viel Geschmack beweist der Maler nicht. Wir haben eben noch die Standard-Techniker-Farben im Malkasten. Wird Zeit, daß wir uns einen neuen POKEn:

```
20 POKE 708,32
30 POKE 709,2
40 POKE 710,64
```

RUN! Sieht das nicht gruselig schön aus? Und probieren Sie auch die Werte 50, 52, 56 - das ersetzt jeden teuren Kamin. 16, 34 und 194 sind aber auch nicht zu verachten. Bedenken Sie, aus 128 Farbtönen können Sie wählen, drei Farben stehen Ihnen zur Verfügung (wenn wir den Hintergrund mal nicht mitzählen), d.h. Sie können in 128*128*128 Farbkombinationen schwelgen, das sind immerhin 2.097.152 Farbdreiklänge. Und die können wiederum auf 128 verschiedenen Hintergründen erscheinen, das macht mehr als 268 Mio. Zusammensetzungen (unbedingt nachrechnen!)

Die Zeile 50 habe ich für Sie freigehalten, damit Sie sich hier an den Hintergrund heranmachen können. Die Farbwirkung der Grafikpunkte ändert sich scheinbar, wenn die Farbe des Hintergrundes wechselt.

Wenn Sie Lust haben anzuschauen, was das Farbspektakel bei feinerer Auflösung bietet, dann:

```
10 GRAPHICS 5+16
20 POKE 708,34
30 POKE 709,100
40 POKE 710,210
60 X=INT(RND(0)*80)
70 Y=INT(RND(0)*48)
80 C=INT(RND(0)*3)
```

```
90 COLOR C+1
100 PLOT X,Y
110 GOTO 60
```

10: schaltet Grafik Nr. 5 ohne Textfenster ein.

20 bis 40: ein neues Farberlebnis!

60: Der größeren Auflösung entsprechend müssen hier Werte von 0 bis 79 möglich gemacht werden.

70: Das gilt natürlich auch für die Zeilen-Werte.

Die übrigen Zeilen bleiben die gleichen. Es wäre also ganz intelligent, wenn sie nicht das ganze Listing neu abtippen, sondern die Werte im vorhergehenden Programm ändern.

Und wenn Sie jetzt auf den Geschmack gekommen sind:

```
10 GRAPHICS 7+16
20 POKE 708,2
30 POKE 709,4
40 POKE 710,10
60 X=INT(RND(0)*160)
70 Y=INT(RND(0)*96)
80 C=INT(RND(0)*3)
90 COLOR C+1
100 PLOT X,Y
110 GOTO 60
```

Hier flimmert GRAPHICS 23 über die Scheibe und Sie sehen, daß die feinere Auflösung nicht immer auch wirkungsvoller ist. Sie meinen, das liegt an den Farben, die ich Ihnen vorprogrammiert habe? Tippen Sie LIST 20,40 und RETURN-Taste und setzen Sie Ihre Farbwerte ein.

Nun, auch dieses Farbgesprenkel sieht so aus wie es ist: zufällig, und das wirkt nicht besonders ästhetisch auf uns. Erst wenn ein gewisses Maß an Harmonie sichtbar wird, empfinden wir ein Wohlgefallen.

Wie sich ein wenig Ordnung auswirken kann, zeigt Ihnen das nächste Programm. Hier wird zwar auch die Veränderung durch zufällige Werte bewirkt, aber das Programm stellt einen Bezug zur senkrechten und waagerechten Mittelachse her. So werden durch einen Zufallswert vier Punkte gefunden, die alle den gleichen Abstand vom Bildzentrum haben. Auf diese Weise kommt ein Gestaltungsprinzip ins Bild. Aber bevor wir weiter über Ästhetik und Computerkunst plaudern, hier erst einmal das Listing:

```
10 GRAPHICS 21
20 CC=0
30 POKE 712,0:POKE 708,16
40 COLOR 1
50 X1=39
60 X2=39
70 X3=39
80 X4=39
90 Y1=23
100 Y2=23
110 Y3=23
120 Y4=23
130 A=INT(RND(0)*7)+5
140 B=INT(RND(0)*7)+5
150 C=INT(RND(0)*2)+1
160 D=INT(RND(0)*2)+1
170 IF C=1 THEN E=1
180 IF C=2 THEN E=-1
190 IF D=1 THEN F=1
200 IF D=2 THEN F=-1
210 G=A*E
220 H=B*F
230 G1=X1+G
240 G2=X2+G
250 G3=X3-G
260 G4=X4-G
270 H1=Y1+H
280 H2=Y2-H
290 H3=Y3-H
300 H4=Y4+H
310 IF G1>79 THEN 130
320 IF G1<0 THEN 130
330 IF G2>79 THEN 130
340 IF G2<0 THEN 130
350 IF G3<0 THEN 130
360 IF G3>79 THEN 130
370 IF G4<0 THEN 130
380 IF G4>79 THEN 130
390 IF H1>47 THEN 140
400 IF H1<0 THEN 140
410 IF H2<0 THEN 140
420 IF H2>47 THEN 140
430 IF H3<0 THEN 140
440 IF H3>47 THEN 140
450 IF H4>47 THEN 140
460 IF H4<0 THEN 140
470 PLOT G1,H1
```

```
480 DRAWTO G2,H2
490 DRAWTO G3,H3
500 DRAWTO G4,H4
510 DRAWTO G1,H1
520 X1=G1
530 X2=G2
540 X3=G3
550 X4=G4
560 Y1=H1
570 Y2=H2
580 Y3=H3
590 Y4=H4
600 PP=RND(0)
610 IF PP<0.8 THEN 130
620 FF=FF+1
630 IF FF=4 THEN FF=0
640 COLOR FF
650 P=RND(0)
660 IF P<0.7 THEN GOTO 130
670 AA=AA+1
680 IF AA>2 THEN AA=0
690 CO=INT(RND(0)*9)*2
700 COL=COL+CO+CC
710 IF COL>255 THEN COL=COL-256
720 POKE 708+AA,COL
730 GOTO 130
```

10: schaltet GRAPHICS 5 ohne Textfenster ein. Damit stehen drei Farben für Grafikpunkte und eine für den Hintergrund zur Verfügung. Die Auflösung beträgt 79 Spalten und 47 Zeilen.

20: Eine numerische Variable auf 0 zu setzen ist überflüssig, diese Zeile erlaubt Ihnen jeodch, verschiedene Anfangswerte für die Farben zu wählen, denn die Variable CC wirkt am Ende des Programms mit auf den Farbwert ein. Experimentieren Sie mit verschiedenen Werten!

30: bestimmt als Hintergrundfarbe (712) Schwarz (0) und für Register 708 den Wert 16 (Standardfarbe 1 = Purpur).

40: COLOR 1 ruft diese Farbe auf.

50 bis 80: Die Spalte 39 wird als Mittelpunkt gewählt, den Variablen X1 bis X3 wird dieser Wert zugeordnet.

90 bis 120: Als mittlere Zeile wird 23 genommen. Die Variablen Y1 bis Y4 bekommen diesen Wert. Damit haben wir vier Variablenpaare (Xn,Yn), mit denen vier Punkte auf dem Bildschirm bestimmt werden können. Der Anfangswert für

diese Variablenpaare ist der Mittelpunkt des Grafikfensters.

130: Hier wird ein Zufallswert zwischen 5 (0+5) und 11 (6+5) gewählt und in der Variablen A aufgehoben. In der nächsten Zeile

140: geschieht das gleiche für B.

150: RaNDomt für C einen Wert von 1 oder 2 und

160: desgleichen für D.

170: fragt die Variable C ab. IF C gleich 1 ist, THEN soll E den Wert 1 erhalten. Für die zweite Möglichkeit ordnet

180: E den Wert -1 zu.

190 und 200: spielen das gleiche Spiel mit F.

210: Hier wird nun die viele Vorarbeit genutzt. Der zufällige Wert A wird mit dem zufälligen Wert E multipliziert. Da E nur +1 oder -1 sein kann, bestimmt E das Vorzeichen für A. Das Ergebnis wird in G gespeichert.

220: So geschieht es auch H, die aus B und F zusammenkomponiert wird.

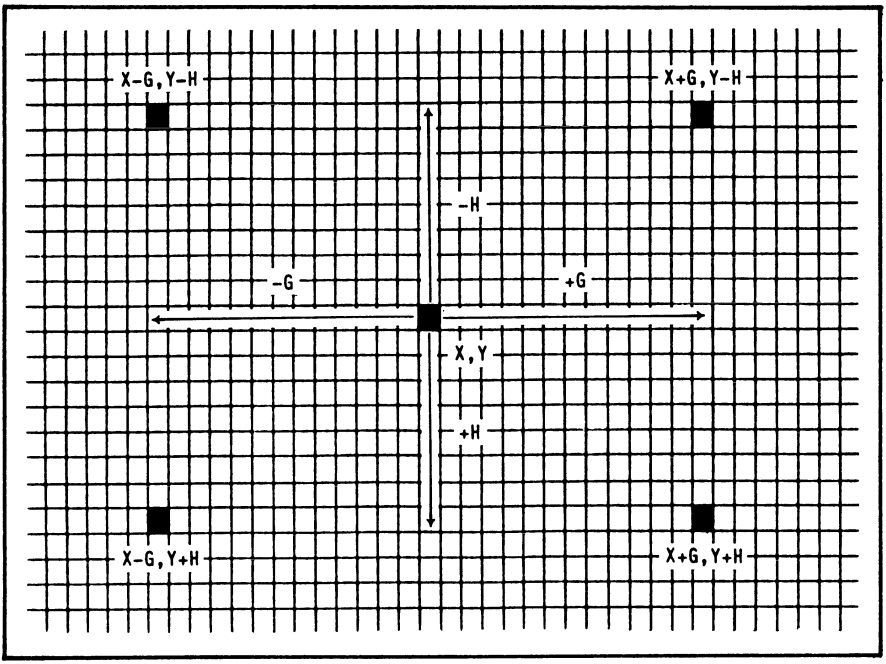
230 bis 260: Die Werte von X1 bis X4 werden durch Addition /Subtraktion von G verändert, die Ergebnisse in den Variablen G1 bis G4 notiert. Und in den Zeilen

270 bis 300: geschieht es ebenso für H1 bis H4.

Die Variablenpaare G1,H1 bis G4,H4 bestimmen vier Punkte im Grafikfenster, die gleiche Entfernungen von den Ursprungskoordinaten X1,Y1 bis X4,Y4 haben, alle zweimal vier Punkte liegen also symmetrisch zur senkrechten und waagerechten Achse.

Die Grafik auf der folgenden Seite zeigt Ihnen eine mögliche Situation an diesem Punkt der Abarbeitung des Programms. Die Koordinatenvariablen X1,Y1 bis X4,Y4 waren anfangs auf den Mittelpunkt (39,23) gesetzt, sie erscheinen als nur ein Punkt, der der Einfachheit halber mit X,Y bezeichnet wurde. Die Pfeile deuten an, wie durch die Variablen G und H die Position dieses Punktes im Grafikfenster verändert wird.

Jetzt muß noch Vorsorge getroffen werden, daß die neu ermittelten Variablen G1 bis G4 und H1 bis H4 keine Werte annehmen können, die größer oder kleiner sind als die zulässigen Bildschirmpositionen (Spalten 0 bis 79 und Zeilen 0 bis 47), damit das Programm nicht mit ERROR- 141 (Cursor-Position unzulässig) abgebrochen wird.



310 bis 380: fragen diese Bedingungen ab und führen ggf. nach Zeile 130 zurück, damit neue Zufallswerte ermittelt werden.

390 bis 460: testen auf gleiche Weise die Variablen H1 bis H4. Hier genügt ein Rücksprung nach 140, denn wenn das Programm bis hierher vorgedrungen ist, muß der für A gefundene Wert im zulässigen Rahmen liegen. (Der Wert für C eigentlich auch. Es wäre also intelligenter gewesen, wenn ich die Zeilen 140 und 150 vertauscht hätte und der Rücksprung hier wäre auf 150 gesetzt.)

Durch diese Rücksprünge wird natürlich Zeit verbraucht. Besonders wenn es der Zufall will, daß mehrmals hintereinander unerwünschte Werte gezogen werden, tritt eine spürbare Verzögerung ein. Wenn das Programm läuft, können Sie beobachten, daß die Veränderungen ruckweise, in unterschiedlichen Intervallen, erfolgen.

Es wäre deshalb sinnvoller, die Programmblöcke anders anzuordnen und alle Zeilen, die sich auf die Ermittlung der Koordinaten beziehen, an den Beginn des Programms zu legen. Damit würde die Rücksprungweite verkürzt und bei

jedem notwendigen Rücksprung würden weniger unnötige Zeilen abgearbeitet. Dieses Problem wird in einem späteren Kapitel noch einmal aufgegriffen. Diese Anordnung dient der besseren Übersicht.

Endlich sind nun die Koordinaten für vier neue Punkte ermittelt. In Zeile

470: wird nun der erste Punkt G1,H1 gePLOTtet, dann

480: wird eine Linie nach G2,H2 gezogen und

490: weiter über G3,H3 nach

500: nach G4,H4.

510: Damit ein Rechteck daraus wird, gehts dann noch zurück nach G1,H1. Aber nicht genug damit! In den Zeilen

520 bis 590: werden die Koordinaten der neu ermittelten Punkte in den Variablen X1 bis X4 und Y1 bis Y4 aufbewahrt. Wenn beim nächsten Programmdurchlauf neue Werte für G1,H1 bis G4,H4 ermittelt werden, dann befinden sich die alten Werte aus diesem Durchlauf in den Variablen X1,Y1 bis X4,Y4. Für dieses Programm ist das ohne weitere Wirkung und könnte unterbleiben. Sie werden aber noch bestaunen können, was sich damit zaubern läßt, wenn wir diese Programmstruktur verändern.

Der nächste Programmblock knöpft sich die Farbe vor. Es ist ja langweilig, nur einen Farbton zu sehen, wenn wir doch drei gleichzeitig darstellen können. Außerdem wäre bei nur einer Farbe bald das ganze Grafikfenster vollgePLOTtet und wir würden gar nichts mehr sehen. In Zeile

600: wird eine Bedingung für den Farbwechsel geschaffen. Der Variablen PP (jetzt fragen Sie mich aber nicht, warum ich die PP genannt habe) wird ein RND-Wert zugeordnet.

610: Wenn dieser Wert kleiner als 0,8 (0.8!) ist, bleibt alles beim alten mit der Farbe. IF aber nicht, THEN findet kein Rücksprung nach 130 statt und es geht weiter mit Zeile

620: wo FF um 1 vergrößert wird.

630: Da wir im weiteren Verlauf für FF nur Werte von 0 bis 3 gebrauchen können, werden größere Werte hier verhindert.

640: taucht den Pinsel in COLOR FF. Wurde der Bildschirm also bislang in COLOR 1 bestrichen, wird ab sofort COLOR 2 verwendet. Die alten Spuren von Farbe 1 bleiben natürlich stehen, bis sie übergepinselt werden.

Wenn es dann das nächste Mal der Zufall will, wird auf

Farbe 3 umgeschaltet. Und weil dann unsere Flimmerleinwand langsam von Farbe trieft, kommt beim nächsten Farbwechsel der Schwamm: Farbe 0 (Hintergrund) schafft optischen Platz für neue Farbanstriche.

Bis hierhin wechseln sich also die vier definierten Farbtöne in zufälligen Intervallen ab. Fügen Sie die Zeile

```
645 GOTO 130
```

ein, um den letzten Programmblock abzuhängen, und Sie können schon mal einen Probelauf auf sich wirken lassen - vorausgesetzt, Sie haben keine Tippfehler gemacht. Sie sehen, daß der Farbwechsel schnell langweilig wird. Unsere nervöse Aufmerksamkeit braucht ständig neue Anregung. Deshalb folgt ein weiterer Geniestreich (645 wieder löschen!):

650: Der Zufall wird noch einmal bemüht und in P verbucht.

660: IF der Wert P kleiner als 0,7 THEN Niete, zurück nach 130. (Sie erinnern sich: Das GOTO hinter THEN darf fortgelassen werden.)

670: Ganz statistisch betrachtet ist der zufällige Wert P in 70% aller (Zu-) Fälle größer als 0,7 und in 30% aller Fälle kleiner. Also etwa bei jedem dritten Mal, bei dem das Programm bis in diese Zeile vordringt, wird AA um 1 erhöht.

680: Da für AA nur Werte von 0 bis 2 erwünscht sind, werder größere Werte hier verhindert.

690: Unter Beihilfe des Zufalls wird ein Wert ermittelt, der als Zuschlag für den Farbwert verwendet werden soll. Da nur gerade Farbwerte wirksam sind, wird mit 2 multipliziert, der Zuschlag kann also gerade Werte von 0 bis 16 annehmen.

700: rechnet CO der Variablen COL hinzu. Hier findet sich auch die Variable CC wieder, die wir in Zeile 20 kennengelernt hatten.

710: sorgt dafür, daß keine unzulässigen Farbwerte durch die fortwährende Addition erreicht werden. Wann immer COL größer als 255 wird (hier könnte auch 254 stehen), wird 256 abgezogen. Hier könnte natürlich auch die Bedingung stehen, COL auf 0 zurückzusetzen, das würde aber zur Folge haben, daß immer bei Schwarz (0) neu begonnen würde.

720: POKEt nun den mühsam zusammenprogrammierten Wert COL in das Farbregister 708+AA. Beim ersten Durchlauf hat AA den Wert 1, der Wert COL landet also in Register 708+1 (=709). Deswegen brauchten wir in Zeile 30 auch nur einen Farbwert für 708 zu programmieren. Beim zweiten Durchlauf

wird der Farbwert in Register 710 verändert. Dann kommt AA mit dem Wert 0, als viertes ist also Register 708 dran usf. Übrigens: Wenn der Farbwert in einem Register umgePOKEt wird, wechseln alle Grafikpunkte auf der Mattscheibe, die ihren Wert aus diesem Register beziehen, schlagartig die Farbe.

730: Puh! Geschafft. Gehe nach 130 zurück und such die Werte für das nächste Rechteck zusammen.

Also jetzt muß ich sie erstmal loben, daß Sie bis hierher durchgehalten haben. War der Happen zu groß?

Auf jeden Fall haben Sie sich jetzt eine Erholung verdient. Spielen sie ein bißchen mit den Werten in der Variablen CC, ändern Sie den Multiplikator in Zeilen 130 und 140 und die Bedingungen für P und PP. Ändern Sie auch Zeile 680, so daß auch die Farbe für den Hintergrund gewechselt wird. Das ist gar nicht so einfach, weil dieser Wert in der Adresse 712 zuhause ist, also nach 710 der übernächsten. Mein Tip: der Maschine ist es völlig schnuppe, wenn zwischendurch ein Wert in 711 abgelegt wird, denn das Register wird hier in diesem Grafikmodus nicht abgefragt und hat auch keine andere Funktion.

So, nun lehnen Sie sich gemütlich in Ihrem Programmiersessel zurück und lassen den meditativen Genuß dieses einmaligen Grafikprogrammes auf sich wirken. Ah.

Aber was ist denn jetzt los! Nach etwa sieben, acht Minuten stillen Genusses ist auf dem Bildschirm plötzlich der Teufel los. Andauernd wechseln die Farben. Ist der Computer kaputt oder der Fernsehkasten oder was. Dahin ist die wohlverdiente Ruhe.

Grund dieser Störung ist die Sorge, die sich ATARI um Ihr kostbares Farbfernsehgerät macht (wie aufmerksam). Wenn eine bestimmte Zeit lang keine Eingabe über die Tastatur erfolgt, wird die bildschirmstrapazierende Helligkeit etwas reduziert und in recht kurzen Intervallen werden die Farbwerte über alle Register gewechselt. Damit soll verhindert werden, daß ein zu lange stehendes Bild in die Röhre einbrennt.

Bei einem Bildschirminhalt, der sich ständig ändert ist das übertriebene Fürsorge. Aber das System ist nun einmal so eingestellt.

Die wenigsten Programme laufen länger als acht Minuten. Wenn Sie aber ein Programm haben, das so lange ohne Eingaben über die Tastatur läuft, z.B. ein Spiel, das über

Joystick gesteuert wird, dann können Sie diesen Bildschirmschoner abschalten. Und das geht so:

Die Adresse 77 ist der Auslöser für den Schongang. Bei jeder Tastatureingabe wird hier der Wert 0 abgelegt. Der Wert in dieser Adresse vergrößert sich von selbst kontinuierlich. Wird irgendeine Taste des Keyboard gedrückt, springt der Wert auf 0 zurück und fängt von vorne an zu zählen. Wird aber ungestört der Wert 128 erreicht und das dauert etwa sieben bis acht Minuten, ergeht der Impuls an die Schonroutine. Glauben Sie nicht?:

```
20 ? PEEK(77);CHR$(127);
30 GOTO 20
```

20: fragt das Register 77 ab und druckt den Wert für uns aus, damit wir sehen können, was darin vorgeht. Die An-Weisung ? CHR$(127); löst einen Sprung zum nächsten Tabulatorstop aus.

30: führt nach 20 zurück.

Wenn Sie diese beiden Zeilen starten, sehen Sie Nullen, nichts als Nullen und nach einer Weile Einsen, lauter Einsen, schließlich Zweien und so fort. Wenn Sie dann irgendeine Taste drücken, schwupp, Nullen, lauter Nullen, und dann wieder Einsen ...

So und nun gehen Sie mal in Ruhe einen Kaffe trinken und schauen in ca. sieben Minuten wieder rein, was dann los ist.

Und hier noch ein: Logischer Verweis.

Möglicherweise gibt es unter den Lesern dieser wichtigen Publikation ein paar Schlaumeier, die sich mit gerunzelter Stirn im obigen Grafikprogramm die Zeilen 170 bis 300 ansehen und dann zu dem klugen Schluß gelangen (den ich in aller Bescheidenheit auch zu ziehen mir erlaubte), daß es ganz überflüssig sei, mit den Variablen E und F Vorzeichen für die Werte G und H zu ermitteln, wenn in den Zeilen 230 bis 300 G und H mal addiert, mal subtrahiert werden. Wird dadurch nicht ein negatives oder positives Vorzeichen aufgehoben? Probieren Sie es aus. (Ich hab mir die Mühe schließlich auch gemacht.) Löschen Sie die Zeilen 150 bis 200 und ändern Sie 210 und 220, indem Sie die Faktoren E und F wegnehmen. Klar, Sie können auch die Zeilen 210 und 220 löschen, aber dann müssen Sie in 230 bis 300 die Gs und Hs durch As und Bs ersetzen und das ist viel mehr Arbeit.

Also RUN.

Was geschieht? Leider ist dies kein Rätselbuch, dann könnte ich die Antwort irgendwo am Schluß verstecken oder ganz für mich behalten. Aber so?

Warum werden nur zwei, drei Rechtecke gezogen und dann geht nichts mehr? Zu den Werten von X1 bis X4 und Y1 bis Y4 wird immer nur etwas hinzugezählt und nach zwei bis drei Additionen sind sie so groß, daß das Programm in den Kontrollzeilen 310 bis 460 immer wieder nach 130 bzw. 140 zurückgeführt wird. Der Rechner ist fleißig am arbeiten, findet nur keine Bedingung mehr, die zu einer Ausgabe führt und auf der Mattscheibe ist absolute Windstille. So kann man reinfallen.

Wenn Sie sich nun von den Programmieranstrengungen erholt haben, dann kann es ja weitergehen? Ich möchte Ihnen nämlich zu gerne demonstrieren, wie verschiedenartige Wirkungen sich aus so einer Programmstruktur herausholen lassen, wenn man nur wenige Kleinigkeiten ändert. (Das ist eine versteckte Aufforderung, doch selbst ein wenig herumzuprobieren.)

Wenn Sie das vorangegangene Listing noch im Computer haben (oder vielleicht abgespeichert auf Band oder Diskette), dann brauchen Sie nur mit dem folgenden Listing zu vergleichen und die neuen Werte einzusetzen:

```
10 GRAPHICS 24
30 POKE 710,0
40 COLOR 1
50 X1=159
60 X2=159
70 X3=159
80 X4=159
90 Y1=95
100 Y2=95
110 Y3=95
120 Y4=95
130 A=INT(RND(0)*5)*9
140 B=INT(RND(0)*5)*9
150 C=INT(RND(0)*2)+1
160 D=INT(RND(0)*2)+1
170 IF C=1 THEN E=1
180 IF C=2 THEN E=-1
190 IF D=1 THEN F=1
200 IF D=2 THEN F=-1
210 G=A*E
```

```
220 H=B*F
230 G1=X1+G
240 G2=X2+G
250 G3=X3-G
260 G4=X4-G
270 H1=Y1+H
280 H2=Y2-H
290 H3=Y3-H
300 H4=Y4+H
310 IF G1>319 THEN 130
320 IF G1<0 THEN 130
330 IF G2>319 THEN 130
340 IF G2<0 THEN 130
350 IF G3<0 THEN 130
360 IF G3>319 THEN 130
370 IF G4<0 THEN 130
380 IF G4>319 THEN 130
390 IF H1>191 THEN 140
400 IF H1<0 THEN 140
410 IF H2<0 THEN 140
420 IF H2>191 THEN 140
430 IF H3<0 THEN 140
440 IF H3>191 THEN 140
450 IF H4>191 THEN 140
460 IF H4<0 THEN 140
470 PLOT G1,H1
480 DRAWTO G2,H2
490 DRAWTO G3,H3
500 DRAWTO G4,H4
510 DRAWTO G1,H1
520 X1=G1
530 X2=G2
540 X3=G3
550 X4=G4
560 Y1=H1
570 Y2=H2
580 Y3=H3
590 Y4=H4
600 CO=CO+1
610 IF CO>50 THEN 630
620 GOTO 130
630 COLOR 0
640 IF CO>100 THEN COLOR 1:CO=0
650 GOTO 130
```

10: fordert GR.8+16 auf, in Aktion zu treten. Da hier die Auflösung anders ist, nämlich 320 mal 192, müssen in den Zeilen

50 bis 120: die Werte geändert werden, die den Bezug zum Mittelpunkt herstellen. Auch in den Zeilen

120 bis 130: ist eine Änderung angebracht. Da die Auflösung sehr fein ist, werden die Zufallswerte mit 9 multipliziert, was zur Folge hat, daß nur jede neunte Spalte bzw. Zeile des Grafikfensters angesprochen werden kann. Hier haben Sie ein breites Betätigungsfeld für eigene Versuche. Ändern Sie den Multiplikator für den RND-Wert (jetzt 5) und/oder den angehängten Multiplikator (jetzt 9). Sie können diesen angehängten Multiplikator aber auch fortfallen lassen und dafür die Bandbreite der möglichen Zufallswerte erhöhen, z. B. auf RND(0)*45. Dadurch würde das starre Schema der Linien etwas mehr aufgelockert.

170 bis 300: bleiben wie gehabt und in

310 bis 460: müssen nur die Grenzwerte der feineren Auflösung angepaßt werden. In

470 bis 510: wird dann wieder das Rechteck gezogen.

520 bis 590: werden noch immer nicht genutzt, aber das kommt bald.

Im letzten Programmteil gibt es erhebliche Änderungen, denn GR.8 erlaubt nur die Wahl eines Farbtones für das Grafikfenster und eine Helligkeit für die Grafikpunkte. Wenn Sie Spaß daran haben, können Sie allerdings im Register 712 dem Bildschirmrand eine eigene Farbe verpassen.

600: Hier finden wir wieder einen der nützlichen Helfer, die treu und brav zählen. Hier heißt der Helfer CO, und er kann nicht mehr, als bei jedem Programmdurchlauf 1 weiterzuzählen. Den Rest besorgt die nächste Anweisung.

610: Wenn CO bis 50 gezählt hat, und das heißt für den Bildschirm, daß er inzwischen 50 gePLOTtete Rechtecke aushalten muß, dann soll das Programm Zeile 620 überspringen und bei 630 weiterarbeiten, denn sonst geht es in

620: zurück nach 130, auf daß der Zufall ein neues Rechteck zeichne.

630: Hier angelangt, wird COLOR 0 eingeschaltet, das ist die Farbe des Hintergrundes. Was das bedeutet, können Sie sehr schön am Bildschirm beobachten: Mattscheibenpflege.

640: Wenn allerdings CO bereits bis 100 gezählt hat, dann wird flugs wieder COLOR 1 eingeschaltet und CO auf 0

gesetzt, auf daß sie wieder von vorne zähle.

650: beendet einen Programmdurchlauf mit dem Rücksprung nach 130.

Die Zeilen 600 bis 640 sorgen also dafür, daß 50 Durchläufe diverse Rechtecke in COLOR 1 gezeichnet werden, dann werden 50 Durchläufe lang Rechtecke mit COLOR 0 gePLOTtet, also optisch wird gelöscht. Danach beginnen wieder 50 Durchläufe, in denen sichtbar gePLOTtet und geDRAWTOt wird und so fort, bis Sie BREAK oder RESET drücken oder den Kasten einfach ausknipsen.

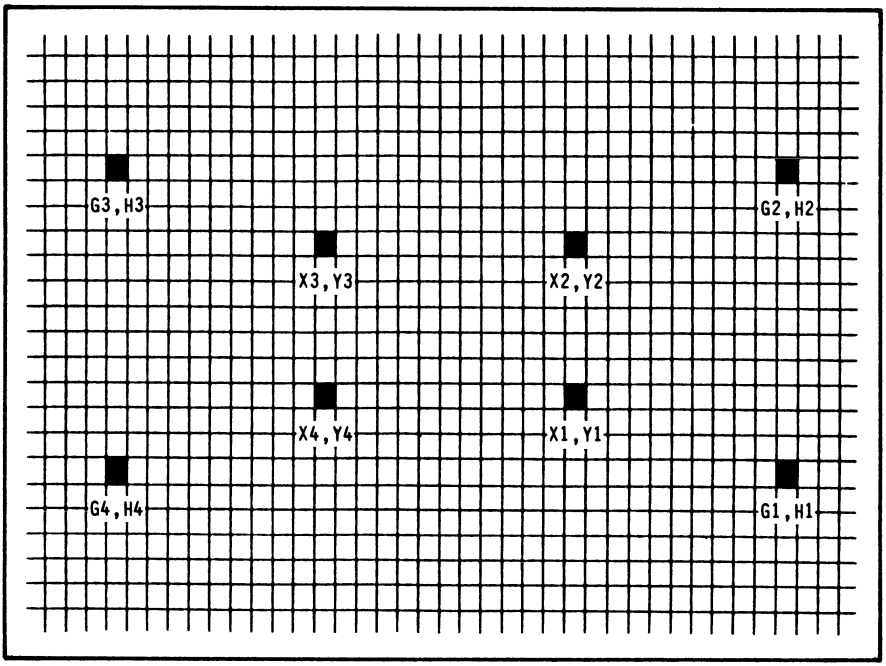
(Aber vor dem Ausschalten darf ich Ihre Aufmerksamkeit noch einmal kurz auf die bläulichen und gelblichen Farbeffekte lenken, die ich mich bereits in Abschnitt 2.1.3 bemüht habe, Ihnen zu erklären.)

Vielleicht sind Sie schon selbst auf die Idee gekommen, daß die vier Punkte nicht zu einem Rechteck verbunden werden müssen. Sie könnten auch nur je zwei waagerecht oder senkrecht zu einer Linie verbinden, aber das bringt nicht viel.

Interessanter ist es schon, jeweils zwei diagonal gegenüberliegende Punkte zu verbinden. Durch die symmetrische Anordnung laufen diese Strecken alle durch das Zentrum. Es entsteht also ein Stern mit verschieden langen Strahlen. Ein Programm für Weihnachten. Sie brauchen dafür nur folgende Zeilen zu ändern:

```
470 PLOT G1,H1
480 DRAWTO G3,H3
490 PLOT G2,H2
500 DRAWTO G4,H4
510 loeschen
RUN und voila!
(es gruesst Ihr Computer
```

Viel mehr Möglichkeiten bieten sich an, wenn wir nun endlich auf die bei jedem Durchlauf aufgehobenen alten Koordinatenwerte zurückgreifen. Uns stehen dann jeweils acht Punkte zur Verfügung, von denen jeweils vier neu erzeugt sind und vier aus dem vorherigen Durchlauf stammen. Sie erinnern sich, daß in den Variablen X1 bis X4 und Y1 bis Y4 die jeweils letzten Bildschirmpositionen notiert wurden und in den Variablen G1 bis G4 und H1 bis H4 die gerade neu ermittelten enthalten sind. Abgesehen von den vielen Kreuz- und Querverbindungen, die zwischen acht Punkten möglich sind, erlauben diese Werte auch das schrittweise Zeichnen ununterbrochener Linien.



Im folgenden Programm wird nun jeweils eine Linie vom zuletzt ermittelten Punkt zur neu gefundenen Koordinate ge-DRAWTOt. Der durch DRAWTO neu erreichte Punkt wird dann der alte Punkt für den nächsten Durchlauf, in dem wieder ein neuer Punkt ermittelt wird, zu dem die Linie fortgezogen wird. Auf diese Weise entspringen bei RUN im Zentrum des Bildschirms vier Linien, die in irren Zacken hin und her über den Bildschirm blitzen und langsam ein Gespinst stricken, daß aber durch die in den Zeilen 230 bis 300 programmierte Beziehung immer hübsch symmetrisch zur senkrechten und zur waagerechten Mittelachse bleibt. Und das macht unserer Wahrnehmung nun mal Spaß. (Bitte bemerken Sie: Symmetrie ist Ordnung. Symmetrie macht uns Spaß. Wie stehts bei Ihnen mit der Ordnung?)

Die Zeilen 10 bis 460 können Sie vom vorherigen Programm unbesehen übernehmen. Dann geht es so weiter:

```
470 PLOT X1,Y1
480 DRAWTO G1,H1
490 PLOT X2,Y2
500 DRAWTO G2,H2
510 PLOT X3,Y3
```

```
520 DRAWTO G3,H3
530 PLOT X4,Y4
540 DRAWTO G4,H4
550 X1=G1
560 X2=G2
570 X3=G3
580 X4=G4
590 Y1=H1
600 Y2=H2
610 Y3=H3
620 Y4=H4
630 CO=CO+1
640 IF CO>100 THEN 660
650 GOTO 130
660 COLOR 0
670 IF CO>220 THEN COLOR 1:CO=0
680 GOTO 130
```

470: PLOTtet den alten Punkt X1,Y1 und

480: zieht eine Linie zum neuen Punkt G1,H1

490 bis 540: verfahren mit den übrigen Koordinaten genauso.

550 bis 670 entsprechen den Zeilen 520 bis 630 der vorhergehenden Programmvariante, nur die Zeilennummern haben sich geändert. In Zeile

670: ist der numerische Wert von 100 auf 220 erhöht worden. Warum wohl?

680: beschließt dieses Programm wieder mit dem Rücksprung.

Das sieht schon ganz gut aus, was? Aber jetzt folgt der krönende Abschluß, wenn wir wieder Farbe ins Spiel bringen. Wir gehen dazu wieder in GR.23, wo unser kreativer Ausflug ja auch begonnen hat. Weil ein Haufen Werte wieder zu ändern ist, hier das vollständige Linsting:

```
10 GRAPHICS 23
20 COL=16
30 POKE 708,COL
40 COLOR 1
50 X1=79
60 X2=79
70 X3=79
80 X4=79
90 Y1=47
100 Y2=47
110 Y3=47
120 Y4=47
```

```
130 A=INT(RND(0)*20)
140 B=INT(RND(0)*20)
150 C=INT(RND(0)*2)+1
160 D=INT(RND(0)*2)+1
170 IF C=1 THEN E=1
180 IF C=2 THEN E=-1
190 IF D=1 THEN F=1
200 IF D=2 THEN F=-1
210 G=A*E
220 H=B*F
230 G1=X1+G
240 G2=X2+G
250 G3=X3-G
260 G4=X4-G
270 H1=Y1+H
280 H2=Y2-H
290 H3=Y3-H
300 H4=Y4+H
310 IF G1>127 THEN 130
320 IF G1<31 THEN 130
330 IF G2>127 THEN 130
340 IF G2<31 THEN 130
350 IF G3<31 THEN 130
360 IF G3>127 THEN 130
370 IF G4<31 THEN 130
380 IF G4>127 THEN 130
390 IF H1>95 THEN 140
400 IF H1<0 THEN 140
410 IF H2<0 THEN 140
420 IF H2>95 THEN 140
430 IF H3<0 THEN 140
440 IF H3>95 THEN 140
450 IF H4>95 THEN 140
460 IF H4<0 THEN 140
470 PLOT X1,Y1
480 DRAWTO G1,H1
490 PLOT X2,Y2
500 DRAWTO G2,H2
510 PLOT X3,Y3
520 DRAWTO G3,H3
530 PLOT X4,Y4
540 DRAWTO G4,H4
550 G0=RND(0)
560 IF G0>0.4 THEN 630
570 C0=INT(RND(0)*29)*2
580 COL=COL+C0
590 IF COL>255 THEN COL=COL-256
```

```
600 POKE 708+P,COL
610 P=P+1
620 IF P>2 THEN P=0
630 X1=G1
640 X2=G2
650 X3=G3
660 X4=G4
670 Y1=H1
680 Y2=H2
690 Y3=H3
700 Y4=H4
710 COLOR P+1
720 FOR Z=0 TO 100
730 NEXT Z
740 GOTO 130
```

10: schaltet Grafik 7 ohne Textfenster ein

20: setzt die Variable COL auf 16 und

30: POKEt den Wert COL in das Register für COLOR 1 (708)

40: schaltet COLOR 1 ein.

50 bis 120: enthalten der gröberen Auflösung wegen die numerischen Werte 79 bzw. 47. Dann folgen in

130 bis 460: die bekannten Kommandos mit entsprechend angepaßten Werten. In

470 bis 540: werden die gleichen PLOT- und DRAWTO-Befehle ausgeführt wie im vorherigen Programm. Der Farbwechsel steht diesmal an einer anderen Stelle im Programmverlauf, aber das ist nicht weiter von Bedeutung.

550: kreiert einen Zufallswert für GO

560: überprüft, wie GO ausgefallen ist und leitet in 60% aller Fälle, nämlich immer wenn GO größer als 0,4 ausfällt und das ist für 60 von 100 Fällen zu erwarten, nach 630 über, wo die bekannte Aufbewahrung der alten Koordinaten beginnt.

570: Wenn die Bedingung aus 560 aber nicht zutrifft, dann bleibt der Sprung aus und es geht hier weiter, indem ein Zufallswert für CO ermittelt wird, der zwischen 0 und 56 liegt, aber nur gerade Zahlen annehmen kann, wie Sie sicher inzwischen schon selbst erkennen. Um diesen Zufallswert CO wird nun in

580: der Wert von COL vergrößert.

590: sorgt in der schon erklärten Form dafür, daß nur

Werte zustande kommen, die als Farbwerte von ATARI akzeptiert werden, also kleiner als 255 und positiv sind.

600: Hier wird der neue Wert COL in das Register 708 ge-POKEt. In dieser Zeile steht zwar 708+P, aber beim ersten Durchlauf ist P noch 0. Es wird erst in Zeile

610: um den Wert 1 erhöht, so daß beim nächsten Durchlauf das folgende Register bedacht werden kann. Da bekanntlich nur drei Farbregister in der gewählten Betriebsart zur Verfügung stehen, sorgt Zeile

620: dafür, daß P keinen größeren Wert als 2 annehmen kann, aber das habe ich ja schon erklärt.

630 bis 700: bewahren die alten Bildschirmkoordinaten auf.

710: wechselt die Farbe. COLOR-Werte 0 bis 3 können aufgerufen werden. P hat wechselnde Werte zwischen 0 und 2. Deshalb wird in dieser Zeile COLOR P+1 aufgerufen (also 1 bis 3). COLOR 0, der Hintergrund, wird in dieser Version nicht bemüht. Aber das können Sie ja ändern.

Weil das Programm für meinen Geschmack etwas hastig läuft, habe ich in

720 und 730: eine Pause programmiert.

740: Rücksprung.

Ich finde, mit dieser Schöpfung können wir uns schon sehenlassen (?). Versenken Sie sich jetzt noch ein wenig in die Abbildung auf Seite 117. Das Programm stellt uns Werte für acht Bildschirmpositionen zur Verfügung. Damit läßt sich eine Menge anfangen. Sie müssen ja nicht immer die neuen mit den alten Punkten verbinden. Viele andere Variationen sind denkbar. Man könnte sogar auf die Idee kommen, vom Zufall bestimmen zu lassen, welcher Punkt im jeweiligen Durchlauf mit welchem anderen verbunden wird.

Sehen Sie sich die Abbildung in Ruhe eine Weile an. Jetzt bilden sich in Ihrer Vorstellung immer neue Linienkonfigurationen, neue Strukturen. Und alle diese Anordnungen sind mit einfachen PLOTs und DRAWTOs zu verwirklichen.

Jetzt ist Ihre Kreativität gefragt.

3.3 Berechnend

Als versierter BASIC-Programmierer sind Sie der Willkür des Zufalls nicht hilflos ausgeliefert. Die endlose Folge RaNDomisierter Werte kann auch lediglich als Motor genutzt werden, um definierte, also geordnete Vorgänge auszulösen.

Das Geschehen in der Natur ist ja ganz ähnlich. Da haben wir auf der einen Seite das wuselige Durcheinander von unzähligen Bewegungen und Veränderungen (Chaos) und auf der anderen Seite gibt es (Natur) Gesetze, die nur unter ganz klar umrissenen Voraussetzungen Ergebnisse ermöglichen.

Stellen Sie sich einen großen Baum vor. Im Herbst bringt er vielleicht 500 oder mehr Früchte hervor und in jeder einzelnen Frucht sind vielleicht ein Dutzend Samenkörner enthalten. Diese Samen werden nun dem Zufall ausgesetzt: Von Tieren (und Menschen) gefressen und verschleppt, vom Wind verweht und damit im statistischen Sinne gleichmäßig über die Umgebung des Baumes verteilt. Aber nur wenn ein Samenkorn ganz exakt definierte Bedingungen von Boden, Feuchtigkeit, Licht und Wärme vorfindet, kann es Wurzeln schlagen und vielleicht zu einem neuen Baum heranreifen.

In ganz ähnlichem Sinn ist das folgende Programm angelegt. Im ersten Teil wird das Grafikfenster mit Punkten zufälliger Farbe gefüllt. Im zweiten Programmteil gibt der Zufall den Motor (Energie, Bewegung) ab, während im Programm die Gesetze (Intelligenz, Richtung) gespeichert sind:

```
100 GRAPHICS 19
110 POKE 708,26
120 POKE 709,54
130 POKE 710,74
140 FOR Y=0 TO 23
150 FOR X=0 TO 39
160 C=INT(RND(0)*3)+1
170 COLOR C
180 PLOT X,Y
190 NEXT X
200 NEXT Y
210 A=INT(RND(0)*40)
220 B=INT(RND(0)*24)
```

```
230 LOCATE A,B,F
240 COLOR F
250 PLOT A,0
260 DRAWTO A,23
270 PLOT 0,B
280 DRAWTO 39,B
290 FOR PAUSE=0 TO 30
300 NEXT PAUSE
310 GOTO 210
```

100 bis 130: schalten die Grafik ein und bestimmen Farbwerte für die Grafikpunkte. Der Hintergrund (712) bleibt unverändert (Schwarz).

140 bis 200: Eine Schleife in der Schleife. Erst wird ein Y-Wert genommen und für alle X-Werte von 0 bis 39 (Spalten) per RND ein COLOR-Wert ermittelt und gePLOTtet. Wenn die X-Schleife abgearbeitet ist, kommt der nächste Wert der Y-Schleife an die Reihe, worauf die X-Schleife wieder ganz abgearbeitet werden muß. Auf diese Weise werden 24 Zeilen mit 39 Grafikpunkten vollgePLOTtet.

Das war das Vorspiel. Das eigentliche Ereignis beginnt in Zeile

200: Ein zufälliger Wert A für eine Spalte und

210: ein zufälliger Wert B für eine Zeile werden ermittelt.

230: Lokalisiert die Bildschirmposition A,B und stellt ihren Wert fest. Wird der LOCATE-Befehl in einer Text-Betriebsart (also GR.0, 1, 2, 12 oder 13) verwendet, ist der gefundene Wert der ATASCII-Wert des Zeichens an der bestimmten Stelle, bei den übrigen Grafikgängen findet LOCATE den COLOR-Wert des Grafikpunktes und füllt diesen Wert in die angegebene Variable. Der LOCATE-Befehl hat also folgendes Format: LOCATE s,z,n - s bezeichnet die Spalte, z die Zeile und n ist eine numerische Variable, die den gefundenen Wert aufnimmt. Spalten und/oder Zeilen können auch durch Variablen oder zu berechnende Ausdrücke ersetzt werden.

Im vorliegenden Programmbeispiel wird der COLOR-Wert des Grafikpunktes A,B in der Variablen F gespeichert.

240: schaltet genau die Farbe des zufällig gefundenen Punktes ein.

250 und 260: ziehen in der Spalte (X-Wert) A eine Linie von oben nach unten (Zeile 0 bis 23).

270 und 280: ziehen in der Zeile (Y-Wert) B eine Linie über alle Spalten (X-Werte 0 bis 39).

Es folgt eine kleine Pause. Das Programm arbeitet sonst so schnell, daß es uns nur vor den Augen flimmert. (Sie können die Pause ja mal verkürzen oder ganz herausnehmen, wenn Sie unbedingt alles ausprobieren müssen.)

Was geschieht auf dem Bildschirm?

Ausgangspunkt ist ein buntes Durcheinander von Grafikpunkten in drei Farben. Statistisch müßte jede Farbe etwa ein Drittel der Gesamtfläche bedecken. Die Farben sind also gleichmäßig verteilt, haben Chancengleichheit.

Jetzt kommt das Schicksal (RND) ins Spiel. Das Programm wählt willkürlich einen Punkt aus und durch diesen Punkt hindurch und in seiner Farbe werden ein senkrechter und ein waagerechter Balken über die volle Größe des Grafikfensters gePLOTtet. Die Farbe dieses Grafikpunktes bedeckt nun eine bedeutend größere Fläche. Im allerersten Durchgang gewinnt die glückliche Partei 39 Grafikpunkte in der Zeile und 23 in der Spalte, von denen etwa ein Drittel schon der eigenen Farbe angehören. Damit ist natürlich die Wahrscheinlichkeit größer geworden, daß im nächsten Durchlauf wieder das Los auf einen Punkt dieser Farbe fällt.

So entsteht ein Gerangel unter den Farben. Nach und nach werden sich kleinere Vorteile für einen Farbwert so angesammelt haben, daß er ein deutliches Übergewicht erkämpft hat. Aber je mehr Fläche eine Farbe schon bedeckt, desto weniger gewinnt sie im Erfolgsfall hinzu. Deshalb haben auch die Zurückliegenden immer noch eine Chance und manchmal kommt es sogar zu einem dramatischen Machtwechsel.

Eine der drei Farben wird relativ schnell zwischen den Fronten aufgerieben. Ist der letzte Grafikpunkt einer Farbe vernichtet, ist diese Partei aus dem Spiel. Das Programm gibt ihr keine Gelegenheit, wieder einzusteigen, denn LOCATE kann ja den entsprechenden COLOR-Befehl nicht mehr finden.

Der Kampf zwischen den beiden verbleibenden Farben ist etwas dramatischer. Aber irgendwann hat sich eine Farbe durchgesetzt. Das Programm läuft dann immer noch weiter, auf dem Bildschirm ist aber keine Veränderung mehr zu sehen.

Programmieren Sie andere (Natur-) Gesetze!

Wie ich schon eingangs erläutert habe, ist dieses Programm durchaus ein (natürlich sehr stark vereinfachtes) Modell für natürliche Vorgänge. Allerdings ist das "Naturgesetz" in diesem Fall sehr plump. Deswegen möchte ich Sie mit dem

nächsten Prgramm noch einen kleinen Schritt weiterführen.

Sie haben sicher schon einmal eine große Wiese gesehen. Da wachsen allerlei Pflanzen und es bilden sich Inseln. Wo schon Pflanzen einer Art stehen, sind die Bedingungen für nachwachsende Pflanzen der gleichen Art günstiger, schon weil die Besämungsdichte größer ist.

Unser Modellfall muß aber nicht in der (gar nicht mehr so heilen Welt der) freien Natur angesiedelt sein. Wirtschaftsunternehmen unterliegen ähnlichen Prozessen. Je näher eine Zweigniederlassung am Stammsitz liegt, desto wirksamer kann der Konkurrenz dort begegnet werden. Die Entscheidungswege sind kürzer, die Kosten geringer. Die Gesetze des Wachstums sind überall ähnlich.

Im folgenden Modell ist der Überlebenskampf differenzierter geregelt. Der "Kampfplatz" wird wieder vom Zufall bestimmt. Dann aber ermittelt der Rechner die Verteilung der Kräfte in einem Gebiet von fünf mal fünf Grafikpunkten um den Zufallswert herum. Durch dieses Kräfteverhältnis wird bestimmt, welches Ereignis eintritt. Dabei werden folgende Fälle unterschieden:
Hegemonie. Eine Farbe hat die absolute Vorherschafft, alle 25 Grafikpunkte haben die selbe Farbe. Die Folge ist ein aggressiver Angriff. Es werden ein senkrechter und ein waagerechter Balken durch das gesamte Grafikfenster gezogen.

Dominanz. Eine Farbe hat mehr als die Hälfte, also dreizehn Grafikpunkte. Die Dominierende Farbe erobert das gesamte Quadrat von fünf mal fünf Punkten.

Mehrheit. Eine Farbe hat mehr Punkte als jede andere. Diese Farbe gewinnt ein Kreuz aus fünf mal fünf Grafikpunkten.

Anarchie. Keine Farbe hat eine Mehrheit. Z.B. können je zwölf Grafikpunkte in einer und der 25ste in der dritten Farbe in dem untersuchten Feld liegen. Wenn keine Mehrheit vorliegt, werden die 25 Punkte des Kampfplatzes in zufälligen Farbwerten neu gePLOTtet. Dabei hat die Minderheit einen statistischen Gewinn.

Aber nach diesen theoretischen und entsprechend trockenen Auslassungen sollten Sie sich in den Kampf stürzen, ich meine damit, das Listing abtippen. Das ist diesmal etwas länger. Die vielen Wachstumsgesetze fordern ihre BASIC-Zeilen. Passen Sie also auf, daß Sie keine Tippfehler machen.

(Ich geh' derweil einen Kaffee trinken):

```
100 GRAPHICS 21
110 POKE 708,144
120 POKE 709,246
130 POKE 710,50
140 FOR Y=0 TO 47
150 FOR X=0 TO 79
160 C=INT(RND(0)*3)+1
170 COLOR C
180 PLOT X,Y
190 NEXT X
200 NEXT Y
210 IF PEEK(77)>120 THEN POKE 77,0
220 F1=0:F2=0:F3=0
230 X=INT(RND(0)*76)+2
240 Y=INT(RND(0)*44)+2
250 FOR A=-2 TO +2
260 FOR B=-2 TO +2
270 LOCATE X+A,Y+B,F
280 IF F=1 THEN F1=F1+1
290 IF F=2 THEN F2=F2+1
300 IF F=3 THEN F3=F3+1
310 NEXT B
320 NEXT A
330 REM HEGEMONIE
340 IF F1=25 THEN COLOR 1:GOTO 390
350 IF F2=25 THEN COLOR 2:GOTO 390
360 IF F3=25 THEN COLOR 3:GOTO 390
370 GOTO 440
380 REM AGGRESSIVER ANGRIFF
390 PLOT X,0
400 DRAWTO X,47
410 PLOT 0,Y
420 DRAWTO 79,Y
430 REM EINE FARBE DOMINIERT
440 IF F1>13 THEN COLOR 1:GOTO 490
450 IF F2>13 THEN COLOR 2:GOTO 490
460 IF F3>13 THEN COLOR 3:GOTO 490
470 GOTO 560
480 REM QUDRAT IN DOMINIERENDER FARBE
490 FOR A=-2 TO +2
500 FOR B=-2 TO +2
510 PLOT X+A,Y+B
520 NEXT B
530 NEXT A
540 GOTO 210
550 REM EINFACHE MEHRHEIT
560 IF (F1>F2) AND (F1>F2)=1 THEN COLOR 1:GOTO 620
```

```
570 IF (F2>F1) AND (F2>F3)=1 THEN COLOR 2:GOTO 620
580 IF (F3>F1) AND (F3>F2)=1 THEN COLOR 3:GOTO 620
590 GOTO 710
600 DRAWTO X+2,Y
610 GOTO 210
620 REM KREUZ FUER EINFACHE MEHRHEIT
630 PLOT X,Y-2
640 DRAWTO X,Y+2
650 PLOT X-2,Y
660 DRAWTO X+2,Y
670 GOTO 210
680 NEXT B
690 NEXT A
700 GOTO 210
710 REM KEINE MEHRHEIT PLOT CHAOS
720 FOR A=-2 TO 2
730 FOR B=-2 TO 2
740 C=INT(RND(0)*3)+1
750 COLOR C
760 PLOT X+A,Y+B
770 NEXT B
780 NEXT A
790 GOTO 210
```

So, da bin ich wieder. Und Sie, haben Sie alles fein abgetippt?

Wenn jetzt alles läuft, d.h. wenn Sie keine Fehler eingebaut haben, können Sie den dramatischen Kampf zwischen Gelb, Rot und Blau am Fernsehschirm verfolgen. Ein Kampf, der jede Fußballweltmeisterschaft erblassen läßt: An den Grenzen spielen sich die tragischen Ereignisse von Aufstieg und Fall großer Reiche ab. Das Innere der Länder wird nur verletzt, wenn hier und da Farblanzen in alle vier Himmelsrichtungen gestoßen werden. Aber diese Eindringlinge haben im Herzen des fremden Reiches kaum eine Überlebenschance. Doch manchmal gelingen solche Einbrüche auch, besonders wenn sie versprengten Einheiten auf feindlichem Terrain zu Hilfe kommen.

Durch die unterschiedliche Bwertung der Mehrheitsverhältnisse läuft der Kampf in diesem Programm viel differenzierter ab und der allererste Zufallstreffer schafft auch kein so großes Übergewicht.

Falls Sie den Aufbau des Programms nun auch noch verstehen wollen, dann muß ich sie leider mit den nachfolgenden Erläuterungen behelligen:

100 bis 130: Grafik und Farbwahl

140 bis 200: eine zufällige Verteilung von Grafikpunkten aller drei Farben wird gePLOTtet.

210: Der Bildschirmschoner. Wenn in Register 77 ein Wert steht, der größer als 120 ist, dann wird auf 0 zurückgesetzt. Hier könnte auch jeder andere Wert stehen, wenn nur verhindert wird, daß der Wert 128 in diesem Register erreicht wird.

220: setzt die Variablen F1, F2 und F3 auf 0, die im Verlauf des Programmes die Jury abgeben werden.

230 und 240: erwählen die zufälligen Koordinaten des Kampfplatzes und in

230 bis 320: findet dann die Abstimmung statt. Die FOR-NEXT-Schleife in der FOR-NEXT-Schleife sorgt dafür, daß um den Zufallstreffer X,Y herum ein Feld von fünf mal fünf Grafikpunkten abgetastet wird.

270: LOCATEt jeden einzelnen dieser 25 Punkte und registriert seinen Wert in der Varaiblen F. Jetzt treten die Schiedsrichter auf:

280: Wenn der lokalisierte Wert 1 (also COLOR 1) ist, zählt Jurymitglied F1 einen Punkt.

290: Für COLOR 2 zählt Kampfrichter 2 und

300: Schiedsrichter F3 zählt die lokalisierten 3er Werte. Danach werden die Abstimmungsergebnisse ausgewertet und die entsprechenden Weisungen erteilt.

330: eröffnet den Hegemonie-Block. Dieses Ergebnis ist leicht festzustellen. Wenn einer der drei Kampfrichter bis 25 zählen konnte, können die beiden anderen nichts abbekommen haben. Es wird dann die Farbe eingeschaltet, die Fn gezählt hat, nämlich COLOR n.

340 bis 360: fragen die Jury ab, und wenn ein Mitglied den Wert 25 erreicht hat, wird seine Farbe eingeschaltet und nach 390 verzweigt. Damit wird die Zeile

370: übersprungen. Hat keiner der Kampfrichter die Hegemonie-Bedingung anzumelden, gelangt das Programm hierher und erhält den Auftrag, bei Zeile 440 weiterzumachen. Damit wird der folgende Programmblock übersprungen, in dem ein Hegemonie-Ergebnis ausgeführt wird.

390 bis 420: starten den aggressiven Angriff, indem sie ein Kreuz durch das gesamte Grafikfenster PLOTten.

425: gibt es gar nicht, aber hier müßte eigentlich der Rücksprung nach 210 verordnet werden, an den Beginn des nächsten Durchlaufs. Da die Zeile 425 fehlt, müßte das Programm eigentlich nach vielen unzutreffenden Bedingungen und etlichen GOTO-Sprüngen schließlich bei Zeile 700 ankommen und dort unbehelligt Chaos stiften. Also ohne Zeile 425 müßte nach jedem aggressiven Angriff im Zentrum dieses Angriffs Chaos gePLOTtet werden. Und weil Zeile 425 wirklich nicht vorhanden ist, frage ich mich (und Sie), warum das nicht geschieht. Bitte überzeugen Sie sich selbst davon, und lassen sie das Programm laufen. Vielleicht entdecken Sie den Grund dafür*. Dann schreiben Sie mir bitte, ich möchte es auch gerne wissen.

430: Auch das Dominieren einer Farbe ist leicht zu ermitteln. Wenn 13 von 25 Grafikpunkten die gleiche Farbe haben, so ist das die absolute Mehrheit. (*dies gilt natürlich auch für jeden größeren Wert!). Also wenn ein Wert größer als 12 (*13 bis 25, dämmerts noch nicht?), liegt Dominanz vor.

440 bis 460: fragen diese Dominanz ab und springen nach 490 in den Ausführungsblock, um Zeile

470: zu überspringen, die direkt zur Auszählung der einfachen Mehrheit überleitet.

480: Dominanz sollte mit einem Quadrat von fünf mal fünf Grafikpunkten belohnt werden. Das geschieht in

490 bis 530: in der vertrauten Weise mit einer verschachtelten FOR-NEXT-Schleife.

540: führt zum nächsten Durchlauf zurück (hier habe ich dran gedacht!)

550: Ja mit der Demokratie ist das so eine Sache. Die einfache Mehrheit ist nicht so leicht festzustellen. Zum Glück sind bei diesem Spiel keine Koalitonen zugelassen. (Nicht daß sie denken, ich könnte das nicht programmieren, aber diese Arbeit!), deswegen bedeutet dann "einfache Mehrheit", daß die Farbe 1 mehr Felder als Farbe 2 bedecken muß und unabhängig davon auch mehr als 3. Nennen wir den Zähler beim Namen, so heißt das, F1 muß größer sein als F2 und F1 muß größer sein als F3. Wir haben also zwei Bedingungen, die beide erfüllt sein müssen, bevor wir weitere Schritte einleiten können. Nun läßt das ATARI-Basic ja zu, IF-THEN-Bedingungen zu verketten. Wir könnten also schreiben:

```
560 IF F1>F2 THEN IF F1>F3 THEN COLOR 1:GOTO 620
```

Leider hat bereits in der Mitte des 19. Jahrhunderts (1815-1864) der englische Mathematiker und Logiker Georg Boole

eine Algebra entwickelt, mit der solche Bedingungen viel einfacher abgefragt werden können. Deswegen sind die Booleschen Operatoren ein Teil von BASIC und deswegen müssen wir uns damit herumschlagen. Puh.

Die Operatoren heißen (logisches) AND, OR oder NOT. Die Operatoren überprüfen Behauptungen oder Bedingungen, die entweder "wahr" oder "falsch" sein können. Ist die Bedingung wahr, ist das Ergebnis 1, ist es unwahr 0. Eine numerische Variable für sich ist wahr (=1), wenn sie ungleich 0 ist und unwahr (=0), wenn sie 0 ist.

In der Rangfolge der Operatoren stehen die Booleschen ganz hinten, die zu überprüfenden Bedingungen müssen deshalb in Klammern gesetzt werden. Mit den Operatoren können auch mehrere Behauptungen verknüpft werden. Geschieht die Verknüpfung mit AND, müssen beide verknüpften Behauptungen wahr sein, wenn das Ergebnis 1 sein soll. Geschieht die Verknüpfung mit OR, muß nur eine wahr sein, um das Ergebnis 1 hervorzubringen. Auf diese Weise können beliebig viele Behauptungen verknüpft werden. NOT stellt alles auf den Kopf, macht aus unwahr wahr und aus wahr unwahr. Soviel in Kürze über Herrn Boole und seine Segnung!

560: Wir haben es mit zwei Behauptungen zu tun. F1 soll größer sein als F2 AND F1 soll größer sein als F3. Und nur wenn dieser Gesamtausdruck wahr (=1) ist, dann soll COLOR 1 eingeschaltet und nach 620 gesprungen werden.

570: kümmert sich um den COLOR-Wert 2. IF F2 größer F1 AND F2 größer F3 THEN Farbe 2 einschalten und nach 620.

580: Und noch einmal: WENN der Ausdruck in der ersten Klammer wahr (=1) ist UND der Ausdruck in der zweiten Klammer wahr (=1) ist DANN Farbe 3 und GEHNACH 620.

590: Durch den Sprung nach 620 würde diese Zeile übergangen. Ist aber keine der vorstehenden Bedingungen erfüllt, dann darf der Programmblock mit den entsprechenden PLOT-Anweisungen nicht bearbeitet werden. Und darüber wacht diese Zeile.

600 und 610: habe ich stehenlassen, um Ihnen zu zeigen (und das ist doch nach der Booleschen Zumutung eine angenehme Abwechselung), wie sich so Fehler in ein Programm einschleichen. 600 und 610 haben inzwischen die Zeilennummern 660 und 670. Ich habe sie beim Schreiben des Programms umnumeriert - und vergessen zu löschen. Da stehen Sie nun, treu und brav, und richten zum Glück kein Unheil an, weil sie auf jeden Fall übersprungen werden. Datenmüll.

Wäre ich bei der Arbeit an diesem Buch das Programm nicht noch einmal Zeile für Zeile durchgegangen, hätte ich diese Hinterbliebenen wohl nie bemerkt, denn das Programm läuft ja auch mit ihnen fehlerfrei. Zwei unnütze Zeilen in einem BASIC-Programm, was macht das schon. Stellen Sie sich aber eine ganze Computer-Gesellschaft vor und die Unmengen herrenloser Informationen, die durch die Datennetze um den Globus geistern. Datenabfälle, die nicht beseitigt werden, weil sie nicht zu "sehen" sind.

Sie sehen an diesem Beispiel, wie dezent sich Fehler in einem Programm verstecken können. Wenn nicht alle Sprungbefehle über 600 und 610 hinweggingen, dann würden sich diese beiden Störenfriede unangenehm bemerkbar machen. Und dann gibt es nur einen Weg: Das ganze Programm Zeile für Zeile durchgehen, so wie es der Rechner auch abarbeiten würde, jedes Kommando überprüfen, jede Variable verfolgen, bis der Tipp- oder Logikfehler aufgespürt ist.

620: Wenn also eine einfache Mehrheit gefunden wurde, dann wird jetzt das kleine Kreuz mit der Balkenlänge von fünf Grafikpunkten gePLOTtet.

630 bis 660: erledigen diese schlichte Arbeit und

700: führt das Programm nach 210, zum nächsten Durchlauf.

710: Jetzt bleibt nur noch die Anarchie übrig. Keine Herrschaft, keine Wahlen. Wenn das Programm bis hierher durchgedrungen ist, dann waren alle anderen Bedingungen nicht erfüllt, also liegt keine Mehrheit in irgendeinem Sinne vor. (Da die Gesamtzahl der überprüften Grafikpunkte, 25, nicht durch 3 teilbar ist, kann dieses Ergebnis nur eintreten, wenn zwei Farben gleich viele Felder bedecken.)

720 bis 780: füllen den Ort der Auseinandersetzung mit Hilfe des Zufalls. Und dann erfolgt in Zeile

790: der Rücksprung nach 210.

Und in dieser Zeile erfolgt der Sprung ins nächste Kapitel.

3.4 Scheinlösung

spätestens seit Erfindung des Kinematographen wissen wir, daß unser Auge mindestens so träge ist wie der Geist. Zwei Bilder verschmelzen zu einer Bewegung, wenn sie nur schnell genug hintereinander gezeigt werden.

Ein Heim-Computer ist in der Regel zu langsam, um bewegte Bilder zu erzeugen. Mehr als eine stehende Grafik fortlaufend zu verändern oder nach und nach aufzubauen, vermag er nicht zu leisten.

Bildinformationen benötigen extrem viel Speicherkapazität. Selbst bei einer am Maßstab unseres Auges gemessen relativ groben Auflösung und der Beschränkung auf eine einfache Farbskala wird schnell das gesamte Elektronengehirnvolumen eines Heim-Computers für ein einziges Bild verschlungen. In BASIC programmiert baut sich das Bild selbst so langsam auf, daß jede Bewegung gefrieren würde.

Trotzdem läßt sich die Trägheit des Auges nutzen, um Bewegung vorzutäuschen. Wie wärs mit einen Tennis-Match:

```
10 POKE 752,1
20 FOR X=0 TO 38
30 POSITION X,20
40 ? " ● "
50 FOR P=0 TO 10
60 NEXT P
70 NEXT X
80 FOR X=38 TO 0 STEP -1
90 POSITION X,20
100 ? " ● "
110 FOR P=0 TO 10
120 NEXT P
130 NEXT X
140 GOTO 20
```

10: schaltet den Cursor aus.

20: Der Schleifenzähler X umfaßt die Werte 0 bis 38, die in

30: die POS.ition bestimmen, an der in Zeile

40: unser Tennisball gePRINTet wird. Das Pseudografik-

Zeichen CHR$(20), einen Punkt, erreichen sie über die Tasten CONTROL und T. Die Leerstellen vor und hinter dem Grafikzeichen löschen diese Schreibstellen. Ein einmal geschriebenes Zeichen oder ein Grafikpunkt verschwindet ja nicht von selbst wieder von der Bildfläche.

50: ist eine kleine Atempause, weil dieses kleine Programm denn doch zu schnell läuft. Verändern Sie die Dauer.

60: beendet die Pause.

70: überprüft den Schleifenzähler X. Wenn die X-Werte von 0 bis 38 abgearbeitet werden, so wird der "Tennisball" von links nach rechts immer in die folgende Spalte gedruckt, wobei die linke Leerstelle jeweils den vorherigen Punkt löscht. Unser Auge läßt uns glauben, der Ball flöge von links nach rechts. Weil alles so schnell geht, sehen wir sogar scheinbar zwei unscharfe Bälle, was den Bewegungseindruck noch unterstreicht. Wenn Sie die Pause auf 100 einstellen, können Sie aber deutlich sehen, wie das Zeichen Spalte um Spalte gedruckt wird.

80: eröffnet den selben Vorgang rückwärts, von rechts nach links, STEP -1. Dadurch entsteht der Eindruck,

90 bis 130: daß der Ball zurückfliegt.

140: führt zurück nach 20, so daß eine endlose Bewegung des Balls von rechts nach links und zurück abläuft.

Durch das fortlaufende Drucken und Löschen kann auch eine streng geordnete Grafik auf Trab gebracht werden. Erste Bewegung kommt ins Bild, wenn sich die Grafik aufbaut. Das kann zwar recht schnell gehen, dauert aber doch sichtlich seine Zeit, je feiner die Auflösung wird und je mehr Punkte zu PLOTten sind.

Wird der Aufbau der Grafik geschickt gelenkt, können damit Bewegungseffekte erzielt werden, ähnlich wie es schon am Tennisball gezeigt worden ist:

```
100 REM FORNEXT1.BIR
110 GRAPHICS 24
120 POKE 712,0
130 POKE 710,0
140 C=C+1
150 IF C=2 THEN C=0
160 COLOR C
170 FOR X=66 TO 256 STEP 5
180 PLOT X,0
190 DRAWTO 322-X,190
```

```
200 NEXT X
210 FOR Y=190 TO 0 STEP -5
220 PLOT 256,190-Y
230 DRAWTO 66,Y
240 NEXT Y
250 GOTO 140
```

100: Was ist REM? REMember ist ein programmierter Denkzettel, eine Zeile, die vom Rechner übergangen wird. In solchen Zeilen kann man als Programmierer Nachrichten hinterlassen. REM-Zeilen können wie jede andere Zeile angesprungen werden (GOTO oder GOSUB). Ich habe hinter REM den Dateinamen vermerkt, unter dem ich das Programm auf Diskette gespeichert habe. Lasse ich das Programm ausdrucken, so habe ich damit quasi eine Überschrift, unter der ich es in meinem Programmarchiv wiederfinden kann.

110: höchste Auflösung (HIRES) ohne Textfenster.

120 und 130: sind für Sie zum Spielen eingerichtet. Hier können Sie nach Belieben andere Farbwerte für das Grafikfenster und den Rand POKEn.

140: Die Variable C zählt von 0 aufsteigend immer 1 weiter,

150: aber nur bis 1, denn wenn sie mühsam 2 erreicht hat, wird sie schon wieder zur 0 zusammengestaucht,

160: damit sie hier den Wert für den COLOR-Befehl abgeben kann. COLOR 1 PLOTtet einen Grafikpunkt, COLOR 0 PLOTtet Hintergrund.

170: definiert den Schleifenzähler X von 66 bis 256 mit einer Schrittweite von 5. Daraus werden die X-PLOTs entwickelt. Bei einer Auflösung von 320 Spalten ergeben die X-Werte 66 bis 256 eine Strecke, die in der Mitte des Grafikfensters liegt und genauso lang ist, wie das Grafikfenster hoch ist.

180: PLOTtet den jeweiligen X-Wert in Zeile (Y-Wert) 0 und

190: zeichnet eine Linie, die in der Zeile 190 liegt, also am unteren Rand und dem gerade gePLOTteten Punkt diagonal gegenüberliegt. Ist X=66, dann ist X-322=256; ist X=256, dann ist 322-X=66.

200: überprüft den Schleifenzähler.

Bis zu diesem Punkt wird folgende Wirkung auf dem Bildschirm erzielt:

Genau in der Mitte des Grafikfensters liegt ein gedachtes Quadrat. Es füllt den Bildschirm in der Höhe aus. Es werden Linien gezogen, die diagonal durch dieses Quadrat laufen.

Die erste Linie verläuft von der Ecke (des Quadrates) oben links nach unten rechts. Die Endpunkte der folgenden Linien rücken dann oben jeweils fünf Bildpunkte nach rechts und unten fünf Bildpunkte nach links, bis als letztes eine Linie gePLOTtet wird, die von oben rechts nach unten links verläuft. Das Quadrat ist jetzt diagonal geviertelt und das obere und untere Viertel sind mit Strahlen in gleichmäßigem Abstand ausgefüllt.

Im nächsten Programmblock sollen nun die Viertel rechts und links auf die gleiche Weise ausgefüllt werden. Es müssen also Y-Werte in Fünferschritten abgearbeitet werden.

210 bis 240: erledigen diese Arbeit. So wie die Werte jetzt gewählt sind, werden die beiden Diagonalen durch das Quadrat zweimal gePLOTtet, nämlich erst in der X-Schleife und dann auch noch in der Y-Schleife. Dadurch entsteht in der scheinbaren Bewegung eine kleine Verzögerung in den Ecken. Wenn Sie den Schleifenzähler für Y auf 185 bis 5 ändern, erhalten Sie eine vollkommen gleichmäßige Bewegung.

250: springt nach 140 zurück, wo C auf den Wert 2 gesetzt wird, der dann in Zeile

150: auf 0 gesetzt wird, so daß in diesem Durchlauf alle Linien in der gleichen Reihenfolge gePLOTtet werden, jedoch in der Farbe des Hintergrundes, d.h. sie verschwinden optisch von der Bildfläche.

So wird das Quadrat im Urzeigersinn mit Linien gefüllt, dann werden die Linien im Urzeigersinn wieder gelöscht und wieder gezeichnet und wieder gelöscht und BREAK!

Diese Abfolge von gezeichneten und wieder gelöschten Linien bildet uns ein, etwas würde sich um den Mittelpunkt des Quadrates drehen. Hätten wir uns die Mühe gemacht, für die PLOTs Endpunkte zu errechnen, die auf einem Kreis um den Mittelpunkt liegen, dann würde der Eindruck eines gleichmäßig rotierenden Ventilators noch viel stärker sein. Aber keine Angst, diese Mühe werden wir uns später noch machen. Die FOR-NEXT-Schleife läßt sich dann allerdings in dieser bequemen Weise nicht mehr nutzen.

Ändern Sie in den Zeilen 170 und 210 die STEPs auf 10, 23.75, 47.5 und 95, natürlich den einen immer positiv und den anderen mit negativem Vorzeichen.

Noch ein Beispiel, wie aus einer FOR-NEXT-Schleife Bewegung herausgeholt werden kann, sollen die beiden folgenden Programme geben. Ein Quadrat in der Mitte des Grafikfensters wird ganz in einer Farbe ausgefüllt. Selbst bei der gröbsten

Auflösung von GR.3 dauert es einen Moment, die 576 zu drucken. Unser Auge kann den Vorgang durchaus beobachten, wenn auch der Eindruck einer fließenden Bewegung entsteht. Im ersten Programmteil füllt sich das Quadrat von oben nach unten, Spalte für Spalte (von links nach rechts). Das sieht fast so aus, als ob ein flinker Maler eine Wand streicht. Wenn die Fläche mit der ersten Farbe lackiert ist, wird die Farbe gewechselt und ein zweiter Anstrich aufgetragen. Jeder, der etwas vom Malerhandwerk versteht, weiß, daß dieser Anstrich quer zum ersten erfolgen muß. Wir lassen den Computer also von links nach rechts Zeile um Zeile pinseln. Dann wird der nächste Farbtopf geöffnet und wieder ein senkrechter Anstrich aufgetragen, danach ein waagerechter usf. So geht es alle 128 ATARI-Farben durch und dann wieder von vorne (mit einem Computer kann mans ja machen):

```
100 REM FOR2NEXT.BIR
110 GRAPHICS 19
120 C=254
130 POKE 708,C
140 POKE 709,C-2
150 COLOR 1
160 FOR X=31 TO 8 STEP -1
170 PLOT X,0
180 DRAWTO X,23
190 NEXT X
200 COLOR 2
210 FOR Y=0 TO 23
220 PLOT 8,Y
230 DRAWTO 31,Y
240 NEXT Y
250 C=C-4
260 IF C<0 THEN C=254
270 GOTO 130
```

120: setzt die Variable C auf 254, den höchstmöglichen Wert für einen Farbton.

130: POKEt diesen Wert für COLOR 1 und

140: eine Nummer heller für COLOR 2.

150 bis 190: pinseln das Quadrat in Farbe 1 spaltenweise von links nach rechts voll.

200 bis 240: pinseln das Quadrat in Farbe 2 zeilenweise von oben nach unten voll.

250: reduziert C um 4, also auf den nächstkleineren Farbwert.

260: setzt C wieder auf 254 bevor es unerwünschterweise

einen negativen Wert annimmt.

Einen fröhlicheren Wechsel in der Farbgebung bietet die folgende Programmversion. Hier werden die COLORs über Variablen aufgerufen und es können Werte aus drei Farbregistern gezogen werden. Hier haben Sie mehr Möglichkeiten zum experimentieren. Verändern Sie die Bedingungen für A und B. Schaffen Sie neue Anweisungen für den Wechsel der Werte in den Farbregistern:

```
100 GRAPHICS 19
110 A=1:B=3
120 POKE 708,C
130 POKE 709,C+2
140 POKE 710,C+4
150 A=A+1:B=B+1
160 IF A=4 THEN A=1
170 IF B=4 THEN B=1
200 COLOR A
210 FOR X=31 TO 8 STEP -1
220 PLOT X,0
230 DRAWTO X,23
240 NEXT X
250 COLOR B
260 FOR Y=0 TO 23
270 PLOT 8,Y
280 DRAWTO 31,Y
290 NEXT Y
300 C=C+2
310 IF C>251 THEN C=0
320 GOTO 120
```

Um diesen Bewegungseffekt zu erzielen, braucht es natürlich keine FOR-NEXT-Schleife. Die erspart uns nur, jeden einzelnen PLOT und DRAWTO zu bestimmen und einzutippen. Diese Erleichterung erkaufen wir uns aber auch mit entsprechenden gestalterischen Beschränkungen. Die Punkte müssen auf einer Geraden liegen. Wie ein Movie-Programm ohne FOR-NEXT aussieht, sehen Sie hier:

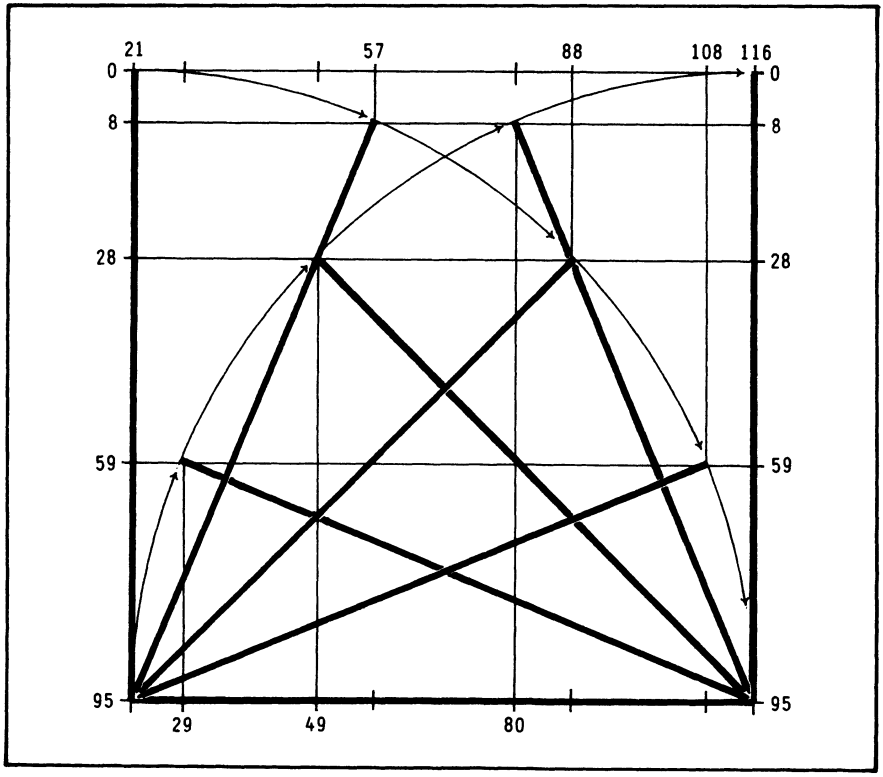
```
100 GRAPHICS 23
110 POKE 708,196
120 POKE 709,186
130 POKE 710,232
150 COLOR 1:PLOT 21,95:DRAWTO 21,0
160 COLOR 1:PLOT 21,95:DRAWTO 57,8
170 COLOR 0:PLOT 21,95:DRAWTO 21,0
180 COLOR 1:PLOT 21,95:DRAWTO 88,28
190 COLOR 0:PLOT 21,95:DRAWTO 57,8
```

```
200 COLOR 1:PLOT 21,95:DRAWTO 108,59
210 COLOR 0:PLOT 21,95:DRAWTO 88,28
220 COLOR 1:PLOT 21,95:DRAWTO 116,95
230 COLOR 0:PLOT 21,95:DRAWTO 108,59
235 FOR P=0 TO 50:NEXT P
240 COLOR 0:PLOT 21,95:DRAWTO 116,95
250 COLOR 1:PLOT 116,95:DRAWTO 21,95
260 COLOR 1:PLOT 116,95:DRAWTO 29,59
270 COLOR 0:PLOT 116,95:DRAWTO 21,95
280 COLOR 1:PLOT 116,95:DRAWTO 49,28
290 COLOR 0:PLOT 116,95:DRAWTO 29,59
300 COLOR 1:PLOT 116,95:DRAWTO 80,8
310 COLOR 0:PLOT 116,95:DRAWTO 49,28
320 COLOR 1:PLOT 116,95:DRAWTO 116,0
330 COLOR 0:PLOT 116,95:DRAWTO 80,8
335 FOR P=0 TO 50:NEXT P
340 COLOR 0:PLOT 116,95:DRAWTO 116,0
350 COLOR 1:PLOT 116,95:DRAWTO 116,0
360 COLOR 1:PLOT 116,95:DRAWTO 80,8
370 COLOR 0:PLOT 116,95:DRAWTO 116,0
380 COLOR 1:PLOT 116,95:DRAWTO 49,28
390 COLOR 0:PLOT 116,95:DRAWTO 80,8
400 COLOR 1:PLOT 116,95:DRAWTO 29,59
410 COLOR 0:PLOT 116,95:DRAWTO 49,28
420 COLOR 1:PLOT 116,95:DRAWTO 21,95
430 COLOR 0:PLOT 116,95:DRAWTO 29,59
435 FOR P=0 TO 50:NEXT P
440 COLOR 0:PLOT 116,95:DRAWTO 21,95
450 COLOR 1:PLOT 21,95:DRAWTO 116,95
460 COLOR 1:PLOT 21,95:DRAWTO 108,59
470 COLOR 0:PLOT 21,95:DRAWTO 116,95
480 COLOR 1:PLOT 21,95:DRAWTO 88,28
490 COLOR 0:PLOT 21,95:DRAWTO 108,59
500 COLOR 1:PLOT 21,95:DRAWTO 57,8
510 COLOR 0:PLOT 21,95:DRAWTO 88,28
520 COLOR 1:PLOT 21,95:DRAWTO 21,0
530 COLOR 0:PLOT 21,95:DRAWTO 57,8
535 FOR P=0 TO 50:NEXT P
540 COLOR 0:PLOT 21,95:DRAWTO 21,0
550 GOTO 150
```

In diesem Film wird folgendes geboten: Links steht ein Stich, der in vier Bewegungsphasen umfällt, bis er flach auf der Erde liegt und sich dann in vier weiteren Schritten wieder aufrichtet, bis er auf der rechten Seite steht. Diese Bewegung läuft vor und zurück.

Beim Umfallen beschreibt der äußere Endpunkt der Linie

einen Kreisbogen, dessen Mittelpunkt der andere Endpunkt der Linie ist. Die PLOT-Werte, die diesen Kreisbogen ergeben, können rechnerisch ermittelt werden. Aber damit will ich Sie erst im nächsten Kapitel belästigen. Die Abbildung zeigt Ihnen, wie die Werte zeichnerisch gefunden werden können.



Die Koordinatenwerte aus dieser Zeichnung tauchen im Listing als PLOT- und DRAWTO-Werte auf:

110 bis 130: POKEn drei Farbwerte. Im Programm wird nur COLOR 1 aufgerufen. Wenn Sie Spaß daran haben, können Sie aber von Zeit zu Zeit die Farbe des Striches wechseln. Sie wissen ja schon, wie man das macht.

150: zeichnet den Strich in der ersten (senkrechten) Stellung.

160: zeichnet den nächsten Strich, der 22,5° geneigt ist. Erst dann wird in Zeile

170: der erste Strich gelöscht, also mit COLOR 0 gePLOTtet. Es sind also immer zwei Striche gleichzeitig zu sehen. Dadurch wird der Bewegungseindruck verstärkt. In den folgenden Zeilen

180 bis 230: wird also immer erst der folgende Strich aufgebaut, bevor der vorhergehende gelöscht wird.

235: erlaubt dem Auge eine kleine Verschnaufpause. Der Strich scheint für einen kleinen Moment waagerecht liegend zu ruhen, bevor er sich in den Zeilen

240 bis 330: zur rechten Seite hin aufrichtet. Nach der Ruhepause können Sie z.B. COLOR 1 durch COLOR 2 ersetzen. Der Strich bekommt dann beim Aufrichten einen anderen Grünton.

335: Nach einer neuerlichen Pause rechts senkrecht stehend, fällt der Strich in

340 bis 430: auf dem gleichen Wege in die Waagerechte zurück, um sich von

440 bis 530: wieder aufzurichten, bis er auf der linken Seite seine Ausgangsstellung wieder erreicht hat.

535: Die Pause.

540: Natürlich darf der ruhende Strich erst nach der Ruhepause gelöscht werden.

550: beginnt das Spiel von vorne.

Statt immer hin und her zu pendeln, könnte der Strich auch aus der rechten Senkrechten nach oben in die Waagerechte klappen, um von dort in die linke Senkrechte zurückzufallen. Dazu müssen Sie natürlich die entsprechenden PLOT-Werte ermitteln. Das Programm wird dann noch einmal so lang.

Es würde dem Bewegungseffekt auch guttun, wenn nicht nur vier Phasen erscheinen würden, sondern z.B. acht. Auch dies ist eine reine Fleißarbeit. Sie müssen nur die entsprechenden Zeilen in das vorhandene Programm einfügen.

Ein weiterer Spaß wäre es, die Farbe des Striches nach jeder Vierteldrehung zu wechseln und so alle möglichen Farbtöne durchzugehen. Also stürzen Sie sich nicht gleich auf das nächste Kapitel, sondern toben Sie sich erst einmal richtig an diesem Programm aus.

3.5 Fliegender Wechsel

Bislang haben wir nur recht einfache Formen in Bewegung versetzt und der Hinweis auf die Kinematographie war wohl etwas hoch gegriffen. Aber in engen Grenzen lassen sich auch schnell wechselnde Bildfolgen erzeugen, wenn auch nur sehr kurze.

Wie aber sollen in BASIC differenziertere Bilder in einem Nu komplett auf der Mattscheibe erscheinen? Das erscheint nicht nur unmöglich, das ist es auch. Der Rechner ist schnell genug, im Augenblick eine Grafik auf die Mattscheibe zu zaubern. BASIC ist der Klotz am Bein, der jeden großen Sprung zum müden Hüpfer macht. Wir müssen also einen Weg finden, die Bilder auf einer Ebene zu wechseln, wo die BASIC-Schnecke nicht kriecht.

Keine Angst, jetzt kommt kein Exkurs in Maschinensprache mit endlosen Kolonnen von Hexadezimalzahlen und so. auch als bescheidener Heim-Programmierer können wir ein wenig in den Schaltkreisen walten. Z.B. durch Umschalten der GRAPHICS-Betriebsart.

Dieses Umschalten geschieht augenblicklich und schneller als unser Auge blickt. Wenn wir ein Bild in GR.5 aufbauen und dann auf GR.7 umschalten, erhalten wir das gleiche Bild durch die feinere Auflösung um die Hälfte kleiner. Weil unser Gehirn sich die Arbeit gerne leicht macht, nehmen wir ein identisches aber kleineres Bild als das gleiche Bild in größerer Entfernung wahr. (das muß man wissen!)

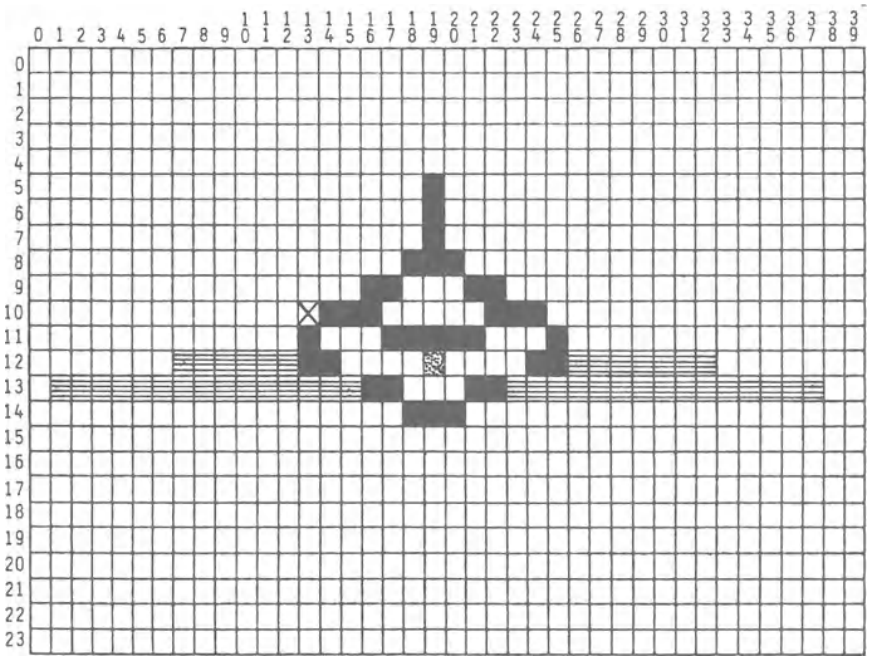
Im folgenden Beispiel kommt ein Vogel geflogen. Er kommt aus der Tiefe des Bildes auf uns zu. Deswegen beginnen wir mit der feinsten Auflösung, dem kleinsten Bild, und schalten über GR.5 auf GR.3. Natürlich muß bei diesem Trick darauf geachtet werden, daß alle PLOT- und DRAWTO-Werte in allen drei Grafikmodi zulässig sind:

```
10 FOR G=2 TO 0 STEP -1
20 GRAPHICS 19+G*2
30 X=10*G
40 Y=6*G
50 COLOR 1
```

```
60 PLOT 19+X,12+Y
70 COLOR 2
80 PLOT 19+X,8+Y
90 DRAWTO 24+X,10+Y
100 DRAWTO 25+X,12+Y
110 DRAWTO 19+X,14+Y
120 DRAWTO 13+X,12+Y
130 DRAWTO 14+X,10+Y
140 DRAWTO 19+X,8+Y
150 DRAWTO 19+X,5+Y
160 PLOT 15+X,10+Y
170 DRAWTO 19+X,11+Y
180 DRAWTO 24+X,10+Y
190 COLOR 3
200 PLOT 26+X,12+Y
210 DRAWTO 37+X,13+Y
220 DRAWTO 23+X,13+Y
230 PLOT 12+X,12+Y
240 DRAWTO 1+X,13+Y
250 DRAWTO 15+X,13+Y
260 FOR P=0 TO 100*G
270 NEXT P
280 NEXT G
290 GRAPHICS 19
300 POKE 712,52
310 FOR Z=0 TO 30
320 NEXT Z
330 POKE 712,0
340 FOR Z=0 TO 500
350 NEXT Z
360 GOTO 10
```

10: eröffnet das Programm mit einem Schleifenzähler, der rückwärts von 2 bis 0 zählt. Dieser Schleifenzähler G bestimmt dann in Zeile

20: die Grafikbetriebsart. Beim ersten Durchlauf hat G den Wert 2, dem mit 2 multipliziert 3+16 (=19) hinzugezählt wird. Oder kurz gesagt, es wird GR.7+16 eingeschaltet. Beim nächsten Durchlauf ist es GR.5+16 und danach GR.3+16. Die Grafikmodi 3, 5 und 7 verfügen jeweils über drei Farben für Grafikpunkte. Die Anzahl der Spalten und Zeilen verdoppelt sich jeweils. Wenn gleich die Grafik gezeichnet wird, dann müssen wir darauf achten, daß alle Koordinatenwerte in den verschiedenen Grafikgängen verarbeitet werden können. Um in GR.5 und GR.7 trotzdem Bildbereiche zu erreichen, die von Werten markiert werden, die bei GR.3 nicht zulässig sind, wird in den Zeilen



Bildschirmskizze des Flugzeuges in Auflösung von GRAFICS 3

30 und 40: eine Verschiebung festgelegt, deren Größe über die Variable G an den Grafikmodus gekoppelt ist; je größer der Grafikmodus, desto größer die Verschiebung. In GR.3 erhalten X und Y die Werte 0. In GR.5 wird X=2*10 und Y=2*6. Lassen Sie das Programm auch ohne diese beiden Zeilen laufen und probieren Sie andere Werte. Das Flugzeug fliegt dann jeweils einen anderen Kurs. Ach ja, der Düsenjäger wird ja erst in den Zeilen

50 bis 250: kreiert. Hier sehen sie, wie mühsam es ist, ein doch vergleichsweise noch recht naives Bild PLOT für PLOT zu umreißen.

260 und 270: schalten eine kleine Augenpause ein, das ist ja für Sie nichts neues. Die Länge der Pause ist über die Variable G an den Grafikgang gekoppelt. Wird der Grafikgang kleiner, die Auflösung gröber, das Bild größer, also kommt das Objekt scheinbar näher, dann wird die Pause kürzer, d.h. die Geschwindigkeit wird scheinbar größer, so wie das bei einem näherkommenden Objekt nun einmal aus-

sieht.

280: überprüft den Schleifenzähler G.

Bevor wir weitermachen, sollten wir uns jetzt erst einmal das Erreichte anschauen. Geben Sie ein

```
285 GOTO 285
```

und starten Sie das Programm. Ja was ist denn das? Der teure Vogel hat ja eine Beule. Bei den feineren Auflösungen fällt das nicht so auf, aber in GR.3 und bei dem stehenden Bild macht sich diese Asymmetrie störend bemerkbar. Da muß ein PLOT- oder DRAWTO-Wert nicht stimmen. Also müssen wir alle Werte mühsam überprüfen. Und wie ich Sie kenne, lesen Sie einfach weiter und die unangenehme Arbeit bleibt wieder mal bei mir hängen.

Aber es ist und ist kein Fehler zu finden. Und doch dieser Ausrutscher. Was ist da los, es ist zum verzweifeln. Also jetzt überprüfen Sie das bitte selbst. Dazu steht ja die Abbildung auf der Vorseite. (Der Schandfleck ist mit einen X markiert.) Alle Werte sind exakt symmetrisch. Und doch erscheint auf der linken Seite ein Grafikpunkt auf 13,10, der rechts nicht zu sehen ist.

Das Problem versteckt sich im DRAWTO-Befehlt. Was denken Sie wohl, was der Rechner bei diesem Kommando macht. Er kennt ja keine Linien sondern nur Bildpunkte. Auf DRAWTO wird ein Vorgang ausgelöst, der zwischen dem gePLOTteten Anfangspunkt und dem Endpunkt alle Bildpunkte errechnet, die dann die Linie ergeben. Liegt diese Linie irgendwie diagonal im Grafikfenster, dann ergibt diese Kalkulation nicht immer ganze Zahlen. Deshalb werden die Werte aus dieser Berechnung im letzten Arbeitsschritt gerundet, die Dezimalstellen fallen unter den Tisch. So wird aus einem Wert 13,1 genauso 13 wie aus 13,9. Da hilft nur sensibles Nachschleifen. In unserem Fall mit

```
125 DRAWTO 13+X,11+Y
```

Sehen Sie, jetzt erfreut das Bild unser Auge, und wir können uns an den Abschluß unseres kleinen Zeichentrickfilms heranmachen. In Zeile

290: wird GR.19 eingeschaltet. Das führt dazu, daß der Bildschirm gelöscht wird. Welcher Grafikmodus hier verwendet wird, ist ziemlich gleichgültig, denn es soll nichts weiter dargestellt werden. Nur GR.0 ist nicht zu empfehlen, weil dann der Cursor oben links in der Ecke jede Illusion zerstört.

300: färbt den ganzen Bildschirm rot. Dadurch sieht es so aus, als ob der Jagdflieger noch näher kommt oder gar explodiert. Sie können ja mal ausprobieren, welche Wirkung an dieser Stelle andere Farben haben. Hellblau (152) vielleicht?

310 und 320: unterbrechen für einen kurzen Moment, bevor

330: die Bildfläche in Schwarz taucht.

340 und 350: gestatten eine längere Verschnaufpause und mit

360: gibt es das Startzeichen für den Abflug der nächsten Maschine.

Wie Sie diesen Stummfilm vertonen können, erfahren Sie in Abschnitt 4.6. Hier bekommen Sie schon mal einen kleinen Vorgeschmack (wiese Geschmack? Gehör!). Ersetzen sie die Pause in Zeilen 340 und 350 durch dieses Geräusch*:

```
340 FOR F=0 TO 90 STEP 0.3:SOUND 0,F,0,15:NEXT F
350 FOR L=15 TO 0 STEP -0.05:SOUND 0,F,0,L:NEXT L
```

Und nun noch der Trick, mit dem bis zu acht verschiedene Bilder in sehr schnellem Wechsel gezeigt werden können. Selbst die Geschwindigkeit eines Propellers ist kein Hindernis. Wir müssen nur Bilder von verschiedenen Stellungen des Propellers hintereinander zeigen.

Bevor dieses beeindruckende Erlebnis elektronische Wirklichkeit werden kann, sind jedoch ein paar Fragen zu klären. Der Propeller beschreibt einen Kreis. Ein Blatt entspricht dem Radius. Wenn wir acht Bewegungsphasen herausgreifen, so dreht sich die Schraube zu jeder nächsten Stellung um 22,5°.

Jetzt bräuchten wir eigentlich nur die acht Propellerblätter zu zeichnen. Aber auf welchen Bildschirmpositionen liegen die Endpunkte? Wir kennen den Mittelpunkt und den Radius, aber damit haben wir noch nicht die X- und Y-Werte für jeden einzelnen Punkt auf der Umlaufbahn.

Natürlich lassen sich diese Werte berechnen, ist doch klar. Und bevor wir uns wieder in die Niederungen einer zeichnerischen Lösung begeben, stellen wir uns endlich dem Problem. Die erste Überlegung geht natürlich in Richtung Pytagoras. Aber mit $a^2+b^2=c^2$ ist dem Problem nicht praktikabel beizukommen. Die Maschine rechnet an den Potenzen und der Wurzel zu lange herum.

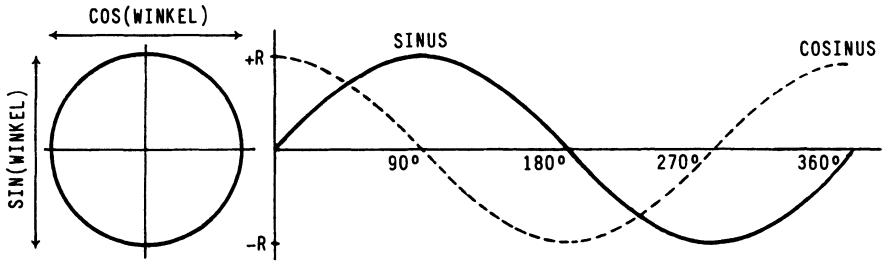
* Dieses Geräusch ist dem Buch "ATARI Sound- und Musik-Buch" von Alfred Görgens entnommen. Es ist in der gleichen Reihe wie dieser Band erschienen.

Es gibt natürlich einen anderen Weg. Der ist zwar im ersten Moment für Nicht-Mathematiker etwas ungewöhnlich, wenn man ihn aber erst verstanden hat sehr zweckmäßig. Da uns der Winkel bekannt ist, können wir den X-Wert mit SIN(Winkel) und den Y-Wert mit COS(Winkel) berechnen:

```
20 FOR WINKEL=0 TO 360 STEP 22.5
30 ? SIN(WINKEL),COS(WINKEL)
40 NEXT WINKEL
```

Dieses kleine Programm geht die sechzehn Winkel durch und errechnet zu jedem den X- und Y-Wert bezogen auf den Radius 1. Diese Werte müssen also noch mit dem tatsächlichen Radius multipliziert werden. Aber schauen wir erst einmal, ob die Formel auch stimmt. (RUN!)

Na? Als gute Mathematikschüler wissen wir beide, daß die SINuswerte von 0° bis 90° auf 1 steigen, dann bis 180° auf 0 fallen, bei 270° weiter bis -1 sinken und mit 360° wieder bei 0 sind; während der COSinus von 1 über 0 und -1 zur 1 zurückfindet.



Leider scheint unser Elektronengehirn grundsätzlich anderer Meinung zu sein! (?) Vielleicht versuchen Sie es einfach noch einmal? Aber nein. Das gleiche Programm wird auch immer die gleichen Ergebnisse erbringen. Da ist der Computer verdammt stur. Auch wenn man felsenfest davon überzeugt ist, daß doch nun wirklich alles so stimmt, wie es da programmiert ist und so laufen und richtige Ergebnisse bringen muß und überhaupt.

Wenn ein Programm nicht funktioniert, dann gibt es beim Computer keine verrußte Zündkerze oder eine falsche Vergasereinstellung, sondern es ist schlicht und einfach ein Fehler im Programm. So unglaublich das auch für uns sein mag. Und dieser Fehler muß gefunden werden. Und das dauert auch schon mal eine ganze Nacht. Und nur zu oft ist

es dann wie in diesem Fall, daß die Programmidee theoretisch schon ganz richtig ist, aber irgendeine Bedienungsvorschrift nicht beachtet wurde. Oder man kannte Sie einfach nicht. In diesem Fall ist es das kleine DEG, was uns zum Programmglück fehlt. DEGree heißt schlicht Grad und meint die Aufteilung des Kreises in 360°. Nichts besonderes? Für den Computer und seine Väter schon. Deren Liebling ist nämlich RADian (auf Hochdeutsch: Bogenmaß), das auf den Einheitskreis mit dem Radius 1 zurückgeht. 180° sind gleichgesetzt mit dem Wert 3,1485926535897undsoweiter (π), und 360° mit dem Doppelten (2π).

Beim Einschalten des Computers oder nach einem Warmstart (RESET) drängt sich immer der RADian in die Schaltkreise. Die zurückhaltenden DEGrees müssen jedesmal extra gebeten werden. Tun wir ihnen also den Gefallen:

```
10 DEG
```

und schon erhalten wir die erwarteten Werte.

Bis jetzt bezieht sich die Funktion aber noch auf den Ursprung (so nennen die Mathematiker die Koordinate 0,0). Die künstliche Intelligenz des (ATARI-) Computers beschränkt sich jedoch auf den vierten Quadranten, also der Ursprung liegt oben links in der Ecke. Um unseren Kreis ins rechte Bild zu setzen, müssen wir die Koordinatenwerte des gewünschten Mittelpunktes jeweils zu den X- und Y-Werten addieren.

Ich hab mir gedacht, wir nehmen für das Propellerprojekt den GRAPHICS 10 (weil es der einzige Modus ist, der uns acht frei definierbare Farben anbietet).

Die Auflösung von GR.10 beträgt 80 Spalten und 192 Zeilen, der Mittelpunkt des Grafikfensters liegt also bei 39,95 und weil der Kreismittelpunkt im Bildmittelpunkt liegen soll, schreiben wir nun:

```
30 ? SIN(WINKEL)+39,COS(WINKEL)+95
```

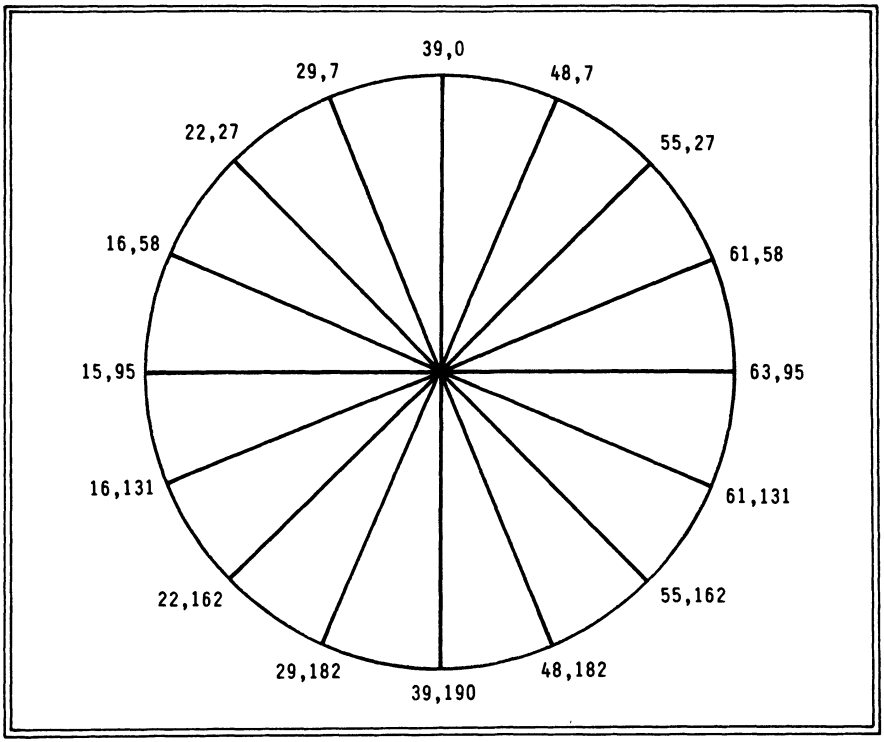
Jetzt fehlt nur noch der Radius. Der Kreis soll natürlich möglichst groß sein. In Y-Richtung kann er einen Durchmesser von 95 Zeilen haben. Da ein Grafikpunkt in GR.10 vier Bildpunkte breit ist, erhalten wir die Anzahl der Grafikpunkte in X-Richtung, die dem Radius entsprechen, wenn wir durch vier teilen (=24). Damit der Radius in Zentimetern gemessen horizontal und vertikal gleich wird, müssen wir also bei dieser unhandlichen Bildauflösung mit den beiden ermittelten Faktoren multiplizieren:

```
30 ? SIN(WINKEL)*24+39,COS(WINKEL)*95+95
```

und weil die PLOTs ohnehin nur ganze Zahlen verarbeiten:

```
30 ? INT(SIN(WINKEL)*24)+39,INT(COS(WINKEL)*95)+95
```

Wenn Sie dieses Programm laufen lassen, spuckt der Rechner die Koordinatenwerte der Propellerspitzen aus (auf den Bildschirm). Diese Werte können Sie dann in das noch zu schreibende Propellerprogramm einsetzen.



Na das war ein Stück Arbeit. Da haben wir uns doch wieder eine kleine Pause verdient, ne? Wenn Sie Lust haben, erweitern Sie das vorstehende Programm um die Zeilen 5, 25, 35 und 50. In Zeile 30 wird der ? durch ein PLOT ersetzt. Das INT können wir uns ruhig sparen, PLOT besorgt das Runden der Werte mit. In Zeile 20 können Sie auch ruhig einen kleineren STEP einsetzen oder den STEP auch ganz wegnehmen. Hier noch einmal das komplette Programm zum abtippen:

```
5 GRAPHICS 26
10 DEG
20 FOR WINKEL=0 TO 360 STEP 22.5
25 COLOR 5
30 PLOT SIN(WINKEL)*24+39,COS(WINKEL)*95+95
35 PLOT 39,95
40 NEXT WINKEL
50 GOTO 50
```

Durch die längliche Form der Grafikpunkte wird der Kreis etwas eierig. Wenn Sie Lust haben, dann arbeiten Sie dieses Kreis-Programm doch auf HIRES um. Aber wenn Sie denken, sie würden dafür mit einem schönen runden Kreis belohnt - weit gefehlt. Ein Kreis in HIRES sieht auf den ersten Blick wirklich schön rund aus. Sie sollten sich aber die Mühe machen, einen Zirkel zu nehmen, und der Erscheinung wissenschaftlich zuleibe rücken. Selbst wenn Sie einen guten Fernsehapparat haben, werden Sie einen deutlichen Unterschied zwischen dem Radius in X- und in Y-Richtung feststellen. Bildschirmzeilen und Punkte haben nämlich keinen gleichen Abstand, Quadrate werden als Rechtecke abgebildet, Kreise als Ovale.

Wenn Sie bei einem gePLOTteten Kreis den Radius in X-Richtung und in Y-Richtung ausmessen und beide Werte durcheinander teilen, dann erhalten Sie einen Meßwert, den Overscan ihres Bildschirmes. Wenn Sie also einen wirklich runden Kreis haben wollen, dann reicht es nicht aus, exakte Werte zu berechnen, sie müssen vor der Ausgabe auf dem Bildschirm den Overscan durch eine entsprechende Multiplikation oder Division ausgleichen. Erst dann bekommen Sie einen wirklich runden Kreis auf die Mattscheibe gelegt. So krumm kann Technik sein.

Jetzt ist aber endlich der Propeller fällig. Also die Idee war, acht verschiedene Stellungen des Propellers schnell hintereinander darzustellen. Im gewählten Grafikgang haben wir für Grafikpunkte acht Farben zur Verfügung, die mit COLOR 1 bis COLOR 8 aufgerufen werden können.

Wir zeichnen je eine dieser acht Propellerstellungen in einer Farbe und setzen alle acht Farbregister auf 0 (=Schwarz). Das ist auch die Farbe des Hintergrundes. (Aber dann sieht man doch gar nichts!) Die Farbregister 705 bis 707 haben ohnehin den Anfangswert 0, die Register 708 bis 711 müssen wir entsprechend umPOKEn. Wenn die acht Propellerstellungen unsichtbar gezeichnet sind, verändern wir in wohlkalkulierter Reihenfolge die einzelnen Farbregister auf einen anderen Farbwert und wieder zurück auf Schwarz. So ist jedes Ein-

zelbild nur für einen kurzen Moment zu sehen und die acht Bilder können fast beliebig schnell hintereinander gezeigt werden:

```
10 GRAPHICS 10
20 POKE 708,0
30 POKE 709,0
40 POKE 710,0
50 POKE 711,0
60 COLOR 8
70 PLOT 39,0
80 DRAWTO 39,190
90 COLOR 7
100 PLOT 48,7
110 DRAWTO 29,182
120 COLOR 6
130 PLOT 55,27
140 DRAWTO 22,162
150 COLOR 5
160 PLOT 61,58
170 DRAWTO 16,131
180 COLOR 4
190 PLOT 63,95
200 DRAWTO 15,95
210 COLOR 3
220 PLOT 61,131
230 DRAWTO 16,58
240 COLOR 2
250 PLOT 55,162
260 DRAWTO 22,27
270 COLOR 1
280 PLOT 48,182
290 DRAWTO 29,7
300 I=I+1
310 IF I>8 THEN I=1
320 POKE 704+I,52
330 POKE 704+I,0
350 GOTO 300
```

10: schaltet GR.10 ein. Ein Textfenster steht hier nicht zur Diskussion.

20 bis 50: setzen die Farbregister 708 bis 711 auf Schwarz.

60: schaltet COLOR 8 ein,

70: PLOTtet den Punkt 39,0 und

80: zieht eine Linie zum gegenüberliegenden Punkt der Kreisbahn.

90 bis 290: bannen auf gleiche Weise die übrigen sieben Propellerlinien auf die Scheibe.

300: Hier wird ein Wert I fortlaufend gezählt und in

310: auf 1 zurückgesetzt, wenn er größer 8 geworden ist.

320: Mit diesem fortzählenden Wert I wird erreicht, daß bei jedem Durchlauf das folgende Farbregister auf einen Farbwert gesetzt wird; ich habe Rosa (52) gewählt.

330: schaltet die Farbe dann gleich wieder auf 0 und

350: führt zu 300 zurück. Der vordere Programmteil muß nicht mehr abgearbeitet werden, die benötigten Linien sind ja bereits alle gePLOTtet.

Na, dann man RUN.

Und wieder einmal die große Enttäuschung? (Wie Sie sehen, sehen Sie nichts, oder besser fast nichts.) Das Umschalten der Farbwerte in den Registern geht tatsächlich so schnell, das ein ähnlicher Effekt wie im Film eintritt, wo bei einer bestimmten Geschwindigkeit die Räder eines Wagens scheinbar rückwärts laufen.

Noch treffender ist der Vergleich mit einem Stroboskoplicht. Eine Lampe blitzt genau immer in dem Moment auf, wo der Propeller eine ganz bestimmte Stellung hat, und er steht augenscheinlich still. Werden die Blitze etwas schneller, erwischt das Licht den Propeller immer etwas früher, und er scheint gar rückwärts zu laufen. Blitzt das Licht etwas langsamer, kommt der Propeller zwischen jedem Aufleuchten etwas mehr als einmal herum, und es sieht so aus, als ob er sich ganz langsam vorwärts bewegen würde. Und ganz ähnlich sieht das bei unserem Programmpropeller aus. Wie empfindlich dieser Effekt reagiert, können Sie feststellen, wenn Sie folgende Zeile in das Programm einfügen:

```
340 FOR P=0 TO 0:NEXT P
```

Rein theoretisch ist diese Zeile wirkungslos. Aber der Rechner braucht doch einen winzigen Moment, um den Befehl zu registrieren und damit ja auch auszuführen. Und dieser unbedeutende Moment von Sekundentausendsteln reicht aus, um den Propeller zum Rotieren zu bringen. Er läuft allerdings etwas tropisch-müde.

Probieren Sie auch eine 1 im Schleifenzähler der Pause - Werte bis 5 ergeben interessante Effekte.

Nachdem Sie nun mit den Pausen ausgiebig herumgespielt

haben, darf ich wohl wieder zum Ernst gemahnen. Denn wenn es auf den ersten Blick auch unsinnig erscheint, möchte ich Sie jetzt bitten (nachdrücklich), das ganze Programm noch einmal neu zu tippen und zwar in der folgenden Form:

```
10 GRAPHICS 10
20 POKE 708,0
30 POKE 709,0
40 POKE 710,0
50 POKE 711,0
60 COLOR 8:PLOT 39,0:DRAWTO 39,190
70 COLOR 7:PLOT 48,7:DRAWTO 29,182
80 COLOR 6:PLOT 55,27:DRAWTO 22,162
90 COLOR 5:PLOT 61,58:DRAWTO 16,131
100 COLOR 4:PLOT 63,95:DRAWTO 15,95
110 COLOR 3:PLOT 61,131:DRAWTO 16,58
120 COLOR 2:PLOT 55,162:DRAWTO 22,27
130 COLOR 1:PLOT 48,182:DRAWTO 29,7
140 I=I+1
150 IF I>8 THEN I=1
160 POKE 704+I,52
170 POKE 704+I,0
180 GOTO 140
```

Dieses Programm ist vom Befehlsinhalt tatsächlich völlig identisch mit dem vorherigen, aber wenn Sie ihm ein RUN verpassen, sehen sie einen wirklichen Propeller! Warum? Es geschieht doch nichts anderes. Der Grafikgang wird eingeschaltet, die Farbwerte werden gePOKEt, die acht Linien in acht COLORs gezogen. Aha! COLOR-, PLOT- und DRAWTO-Befehle stehen in einer Zeile. Aber das kann doch gar keine Auswirkungen haben, denn wenn die PLOTs einmal erledigt sind, kommt das Programm doch gar nicht mehr in diese Zeilen. Oder doch?

Auf jeden Fall ist augenscheinlich, daß diese Programmversion um ein Geringes schneller läuft. Und das kann ja nun doch nur damit zu tun haben, daß einige Befehle in einer BASIC-Zeile zusammengefaßt sind.

Bei den Übungsprogrammen in diesem Buch steht meist nur ein Befehl pro Zeile, weil das übersichtlicher und besser abzuschreiben ist. Tatsächlich läuft ein Programm jedoch um Sekundenbruchteile schneller (und die zählen im Computer-Zeitalter), wenn möglichst viele Befehle in einer Zeile zusammengefaßt sind.

In unserem Propellerbeispiel passiert folgendes. In der

letzten Zeile ergeht die Anweisung zu einem Rücksprung. für einen Computer, der seine Dummheit nur durch Fleiß und Pedanterie ausgleichen kann, bedeutet das, daß er von oben anfängt, jede Zeilennummer mit dem Rücksprungbefehl zu vergleichen, bis er schließlich die gesuchte Zeile ermittel hat. Und da hat er mit der zweiten Programmversion offensichtlich weniger zu tun, weil er weniger Zeilennummern durchzusehen hat.

Würde das Programm also noch schneller laufen, wenn der Rücksprung über noch weniger Zeilen erfolgen würde? Probieren Sie es aus. Der Propeller wird es Ihnen zeigen. Stellen Sie den Block mit der Rücksprungschleife an den Programmbeginn und die Zeilen, die ohnehin nur einmal bearbeitet werden, schieben wir ans Ende:

```
10 GRAPHICS 10
20 GOTO 80
30 I=I+1
40 IF I>8 THEN I=1
50 POKE 704+I,52
60 POKE 704+I,0
70 GOTO 30
80 POKE 708,0
90 POKE 709,0
100 POKE 710,0
110 POKE 711,0
120 COLOR 8:PLOT 39,0:DRAWTO 39,190
130 COLOR 7:PLOT 48,7:DRAWTO 29,182
140 COLOR 6:PLOT 55,27:DRAWTO 22,162
150 COLOR 5:PLOT 61,58:DRAWTO 16,131
160 COLOR 4:PLOT 63,95:DRAWTO 15,95
170 COLOR 3:PLOT 61,131:DRAWTO 16,58
180 COLOR 2:PLOT 55,162:DRAWTO 22,27
190 COLOR 1:PLOT 48,182:DRAWTO 29,7
200 GOTO 30
```

Ein Propellereffekt läßt sich natürlich auch einfacher erzeugen, wie Sie schon in Abschnitt 3.4 erfahren haben, jedenfalls, wenn wir uns mit einem Strich als Rotorblätter zufrieden geben wollen. Bei der hier vorgestellten Methode können aber acht beliebige Bilder nacheinander gezeigt werden. Es ist ganz unerheblich, wie kompliziert diese Bilder sind und wieviel Rechnerzeit es kostet, sie auf dem Bildschirm aufzubauen, denn der Film läuft erst ab, wenn die acht Grafiken längst stehen. Wenn Sie die Mühe nicht scheuen, können Sie also richtige Rotorblätter PLOTten, Sie können aber auch acht Bewegungsphasen einer Comicfigur, eines Gesichtes, eines - viel Spaß!

4 Spiele!

Was denn nun eigentlich Spiel sei, darüber machen sich gelehrte Männer schon lange ernstes Kopfzerbrechen. Man ist sich so ungefähr darin einig, das Spiel als Teil menschlicher Kultur zu betrachten. Eine menschliche Gruppe ist nicht ohne Spiele denkbar. Mit dem Spiel ginge auch das Menschliche verloren. Und gerade bei der Computertechnologie hat sich wieder gezeigt, wie sich der Mensch über das Spiel mit dem Fremden, Neuen vertraut macht. Erst die Computer-Spiele haben eine breitere Öffentlichkeit an den Computer gelockt.

Teil das Lebens ist das Spiel auch gleichzeitig Abbild des Lebens, eine begrenzte Darstellung, vergleichbar mit einer Fotografie. Begrenzung ist ein herausragendes Merkmal von Spiel. Das Spiel braucht immer einen begrenzten Platz, das Spielfeld. Und die Spielregeln gelten auch immer nur für eine begrenzte Zeit, die Spieldauer.

Natürlich werden auch Spieler benötigt. Bei manchen Spielen bewegen sich die Spieler selbst, bei anderen gibt es Figuren, mit denen sich die Spieler identifizieren. Sie tun so, als ob sie die Figur wären und in der Welt des Spielplans lebten.

Der Spieler braucht einen Gegenspieler. Nur aus widerstrebenden Kräften entsteht ein Spiel (das freie Spiel der Kräfte). Ein Spiel allein braucht den Widerstand in einer bestimmten zu vollbringenden Leistung, doch das wirkliche Vergnügen bietet nur ein menschlicher Gegenspieler, der mit der ganzen Vielfältigkeit seiner Persönlichkeit reagiert, der Logik, Gefühl und Intuition ins Spiel bringt und immer wieder für Überraschung sorgt.

Kontrahenten spielen also nicht wirklich gegeneinander. Sie spielen miteinander um die gemeinsame Freude, und notwendigerweise führt der Kräftevergleich dazu, daß einer sich nur durchsetzen kann. Das Spiel macht aber (auch dem Sieger) nur Spaß, wenn nicht von vorne herein mit seinem Sieg zu rechnen war, d.h. alle Spieler müssen eine etwa gleiche Chance haben und notfalls werden schwächeren Spielern Vorgaben eingeräumt, um diese Angleichung zu erreichen.

Computer-, Video-, Tele- oder wie sie immer heißen mögen, sind, in der heute üblichen Form, keine wirklichen Spiele. Sie sind ein billiger Erstazstoff einer Epoche, die dabei ist, das Menschliche wegzurationalisieren.

Die elektronische Unterhaltungsdroge macht abhängig, weil sie sich als Spiel anbietet, aber kein Spiel bietet. Schon wenn das Geldstück in den Schlitz gesteckt wird steht fest, wer am Ende gesiegt hat, nämlich die Maschine. Es fasziniert uns, daß ein totes Gerät scheinbar menschlich reagieren kann. Aber jede Kleinigkeit ist vorprogrammiert, jeder Spielzug, jede Reaktion ist vorab berechnet und alles läuft mechanisch und geistlos ab. Keines dieser Spiele, auch nicht der erfolgreichsten, hat sich länger als ein paar Monate in der Gunst des Publikums gehalten. Der Computer-Spieler sucht in immer neuen Angeboten endlich zu finden, was ihm versprochen wurde: Leben, doch er bekommt einen Roboter.

Selbst um die Schach-Computer ist es nach kurzer Zeit wieder still geworden, weil sie einen menschlichen Gegner eben doch nicht ersetzen können. Und ist es nicht ein trauriges Bild, wie ein Mensch, einsam, gegen einen kleinen Kasten spielt. Ein paar Monate hat uns die technische Möglichkeit fasziniert. Aber das Schachspiel selbst schlägt die Menschheit seit Jahrhunderten in den Bann!

Interessant könnten Computer-Spiele werden, wo die Elektronik das Spielbrett abgibt, eine Spielwiese mit nahezu unbegrenzten Möglichkeiten, auf denen sich menschliche Spieler begegnen können. Die Dimensionen des Spielens könnten damit unendlich erweitert werden. Doch in unserem Jahrhundert der Einsamkeit werden Computer-Spiele gekauft, weil es an menschlichen Spielpartnern mangelt oder die elektronische Sinnenbetäubung verlockender ist, als die Hinwendung zu einem Partner.

Computer-Spiele finde ich auch interessant, wo der Spieler sie selbst programmiert. Das Programmieren ist das Spiel. Die Herausforderung, eine (Spiel-) Idee so zu durchdringen und in allen Aspekten zu verstehen, daß man in der Lage ist, jeden möglichen Zug vorzuprogrammieren. Wenn Sie sich diesem Spaß widmen wollen, haben Sie das richtige Buch in der Hand. Die Spiele, die Sie bald schreiben werden, werden Sie kaum spielen. Wenn das Programm geschrieben ist, dann ist das Spiel beendet. Das ist wie bei der elektrischen Modelleisenbahn. Wenn die Berge modelliert, die Schienen verlegt, die Signale verkabelt und die Landschaft gestaltet ist, ja dann fährt der Zug halt herum. Zuschauen ist kein Spiel.

4.1 Der Spieler

Eines der spannendsten Spiele ist es, ein Spiel zu programmieren. Der Programmierer ist dann der Spieler, der sich auf dem Spielplatz des Computers mit der Logik als Gegner mißt. Aber wie jedes Spiel findet auch dieses sein Ende, wenn das Programm geschrieben ist. Allerdings - Künstler sagen oft, ihre Werke würden nie fertig. Immer gibt es noch eine Idee, etwas besser zu machen, zu verändern, zu probieren, mit den Möglichkeiten zu spielen.

So wird es Ihnen hoffentlich auch mit den Programmen ergehen, die Sie entwerfen und schreiben, wenn Sie dieses Buch ausgelesen endlich beiseite legen.

Als erstes brauchen wir für unser Computer-Spiel eine Spielfigur, die den Spieler darstellt, und wir müssen eine Verbindung zwischen dem Spieler und der Figur herstellen. Bei einem Brettspiel ist das selbstverständlich. Wir nehmen die Figur und setzen sie um. Aber den Spieler auf der Mattscheibe kann man nicht anfassen.

Wir haben also zwei Fragen zu klären. Wie stellen wir den Spieler mit den Mitteln des Computers dar? Und wie kann der Spieler bewegt werden?

4.1.1 Über STICK und Stein

Die naheliegendste Möglichkeit, in das Spielgeschehen einzugreifen, bietet die Tastatur. Das Keyboard ist ja ohnehin das Medium, über das wir die Maschine ansprechen.

Der BASIC-Befehl INPUT (=Eingabe) unterbricht den Ablauf eines Programmes und wartet auf eine Eingabe über die Tastatur. Auf diese Weise können numerische Werte (INPUT n) oder Zeichenketten, Buchstaben, Ziffern, Wörter (INPUT s$) erfolgen.

Der INPUT-Befehl könnte z.B. bei einem Ratespiel eingesetzt werden. Das Programm wählt eine zufällige Zahl (INT(RND(0) *99+1). Dann wird der Spieler aufgefordert, die Zahl zu raten. Ein INPUT liest den eingetippten Versuch in eine Variable und vergleicht diese mit der vom Computer ermittelten Zahl. Ist der INPUT des Benutzers kleiner als die Zufallszahl, kann ein entsprechender Text ausgedruckt werden. Wenn sie größer ist natürlich auch. Dann ist der Spieler wieder an der Reihe, einen Tip abzugeben. Hat er irgendwann endlich die richtige Zahl eingeINPUTtet (oder: INgePUTtet?), also INPUT = Zufallszahl, ist das Spiel beendet.

(Dieses Prograämmchen können Sie selbst schreiben. Wirklich, das können Sie. Oder wollen Sie etwa im Kapitel 5.1 nachschauen?)

Bei diesem Beispiel wird die Tastatur in der gewohnten Weise zur Kommunikation mit dem Rechner genutzt. Jede Taste bedeutet dabei genau das, was auf ihr draufsteht, nämlich A, B, C oder 3, 2, 1. Als Programmierer können wir den Tasten aber auch neue Bedeutungen zuweisen. Der Druck auf Taste N könnte den Spieler ein Feld nach oben rücken lassen, die Taste S den Abbruch des Spiels bewirken. So könnte theoretisch jede einzelne Taste mit einem Spielbefehl belegt werden, oder noch richtiger jedes ATASCII-Zeichen, denn jedem dieser 256 Zeichen ist ein numerischer Wert zugeordnet. Beim Drücken einer Taste wird dieser Wert erzeugt und statt durch diesen Wert ein Zeichen auf den Bildschirm zu bringen, kann er von unserem Programm auch in anderem Sinne verarbeitet werden.

Neben den alpha-numerischen Tasten und den Steuertasten hat jeder Computer noch mehr oder weniger viele Funktionstasten, die dem Programmierer zur freien Verfügung stehen. Bei Ihrem ATARI befinden sich diese Funktionstasten auf der rechten Seite und sind im Hinblick auf den Spieleinsatz mit OPTION (Vorwahl), SELECT (Auswahl) und START (Beginn) beschriftet, was uns aber nicht daran hindern kann, mit dem Druck auf die START-Taste den Abbruch des Spiels auszulösen. Die (SYSTEM) RESET-Taste ist keine frei belegbare Funktionstaste, aber das werden Sie inzwischen schon gemerkt haben, nehme ich an.

Wie das mit den Funktionstasten funktioniert wollen Sie wissen?:

```
10 ? PEEK(53279):GOTO 10
```

Was sehen Sie? Eine endlose Spalte von 7en flimmert über den Bildschirm. Und nun betasten Sie mal nach Herzenslust die Funktionstasten! Nur keine Angst, durch Tastendrücken ist kein Computer klein zu kriegen. Sie dürfen also ruhig zwei Tasten auf einmal drücken oder alle drei.

In der Adresse 53279 wird registriert, welche Funktionstaste /n gedrückt ist/sind. Es können Zahlenwerte von 0 bis 7 entstehen. Mit PEEK kibitzen wir in diese Adresse und ? bringt den aktuellen Wert an den Tag.

Funktionstasten		und OPTION	und SELECT	und START
OPTION	3		1	2
SELECT	5	1		4
START	6	2	4	
	7	keine Taste gedrückt		
	0	alle Tasten gedrückt		

Die verursachten Zahlenwerte können vom Rechner entsprechend der von uns programmierten Anweisungen verarbeitet werden:

```
500 F=PEEK(53279)          Wert aus 53279 in Variable F
600 IF F=3 THEN 100        OPTION gedrückt
700 IF F=2 THEN A=12       OPTION und START gedrückt
800 IF F=6 THEN END        START gedrückt
```

Die Abfrage der Tastatur erfolgt auf ganz ähnliche Weise. Wir müssen dem Rechner nur bedeuten, daß wir die Tastatur abfragen und nicht Zeichen eintippen wollen. Dazu eröffnen wir einen Datenkanal zur Tastatur

```
10 OPEN #1,4,0,"K:"
```

nehmen den Wert der gedrückten Taste in eine Variable

```
20 GET #1,T
30 ? T
```

und schließen den Datenkanal hinter uns

```
40 CLOSE #1
50 GOTO 10
```

10: erOPENt den Datenkanal Nummer 1. Hier könnte auch eine andere Kanalnummer bis 23 stehen, es können also mehrere Datenkanäle nebeneinander geöffnet sein. Die Eingabe einer falschen Kanalnummer führt erst zu einer Fehlermeldung, wenn das Programm läuft: ERROR- 20 AT LINE 10 Der zweite Wert hinter OPEN ist die Kennzahl der Betriebsart (4=Eingabe), eine weitere Kennzahl folgt und zwischen den Anführungsstrichen steht das Gerätekennzeichen. "K:" leitet den Datenkanal zum Keyboard, "D:"iskettenstation, "C:"assettenrekorder, "P:"rinter, "E:"ditor (Textfester), "S:"creen (Grafikfenster)

20: GET pickt eine Information auf und verstaut sie in der angegebenen Variablen. Nachdem in Zeile 10 der Datenkanal zur Tastatur eröffnet wurde, wartet das Programm in dieser Zeile, bis eine Taste gedrückt worden ist. Erst dann wird die nächste Zeile bearbeitet. GET wirkt also ähnlich wie INPUT.

30: Der in die Variable T geGETtete Wert wird ausgePRINTet.

40: Es klingt jetzt vielleicht banal, aber ein eröffneter Datenkanal muß geschlossen werden, bevor er wieder geöffnet werden kann. Soll durch das GET z.B. ein GOTO ausgelöst werden, vergißt man das schon mal schnell. Und dann hat der Rechner den Datenkanal voll und schmollt mit ERROR-. Also erst schön CLOSEn, dann erst

50: zurückspringen.

Ich hätt mir denken können, daß Sie's merken! Das Programm funktioniert, es ist auch sehr hübsch und ordentlich aufgebaut, aber für unseren Zweck hier wäre es auch viel einfacher gegangen. Wir springen in Zeile 40 nach 20 zurück und sparen uns den CLOSE-Befehl ganz, weil das Programm

endlos von 20 nach 40 läuft und nur unterBREAKt werden kann. Ein neuerliches OPEN kann gar nicht erfolgen. Solche Progrmmierstückchen sollte man sich aber erst leisten, wenn man die Befehle auch im Griff hat.

Nun tippen Sie das Fünf-Zeilen-Programm schon ein und lassen es laufen! Was passiert, wenn Sie die Taste CAPS, INVERS oder CONTROL drücken und warum? Und wenn Sie BREAK drücken? Welchen Wert bringt CURSOR HOCH, TAB SET oder ESC?

Wie bei den Funktionstasten lassen sich diese Zahlenwerte auch im Programm verarbeiten (Tasten-Trainer):

```
10 OPEN #1,4,0,"K:"
20 GET #1,T
30 IF T=27 THEN ? "ESC"
40 IF T=32 THEN ? "LEERTASTE"
50 IF T=125 THEN ? "CLEAR"
60 IF T=126 THEN ? "BACK SPACE"
70 IF T=127 THEN ? "TAB"
80 IF T=155 THEN ? "RETURN"
90 IF T=156 THEN ? "DELETE"
100 IF T=157 THEN ? "INSERT Zeile"
110 IF T=158 THEN ? "CLR TAB"
120 IF T=159 THEN ? "TAB SET"
130 IF T=255 THEN ? "INSERT Zeichen"
140 GOTO 20
```

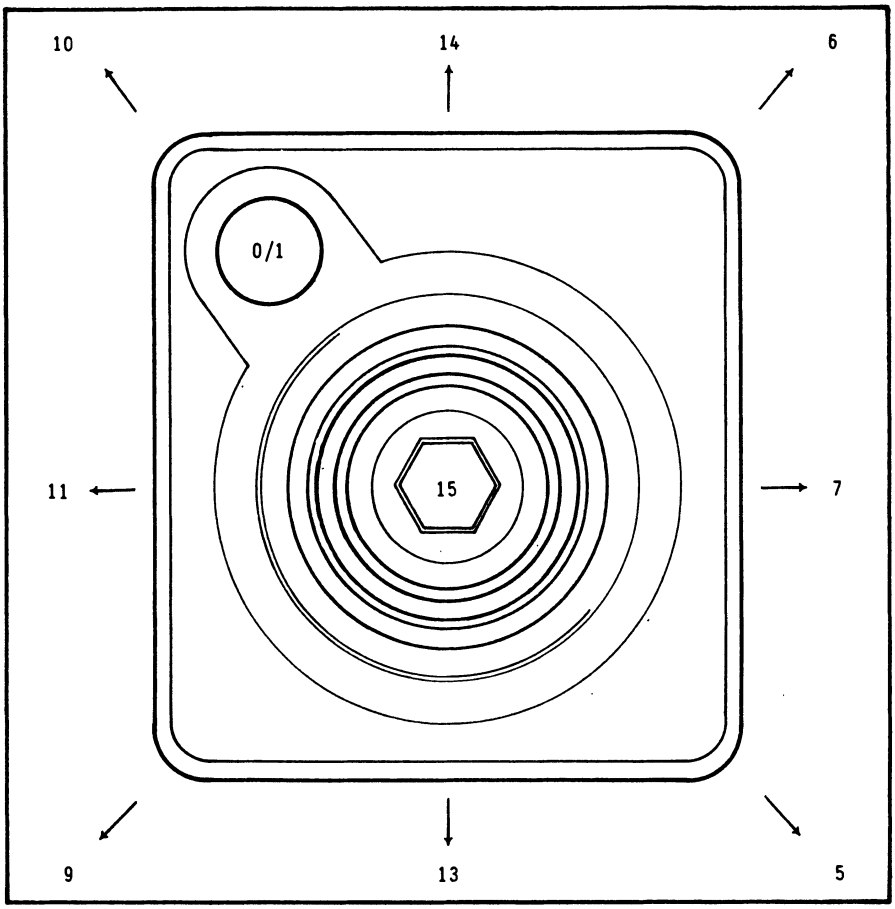
Wenn Sie diesem Programm ein RUN geben, dann können Sie nach Herzenslust die Tasten rings um die Buchstaben herum befingern und auf dem Bildschirm lesen, was diese Taste/n bewirkt/en.

Eine weitere Eingabemöglichkeit ist typisch für Heim-Computer und zeigt die Abstammung vom Telespiel: die Joysticks (ganz freizügig übersetzt: Steuerknüppel).

Der ATARI-Joystick (der auch bei Commodore und Sinclair verwendet wird) hat inklusive Ruhestellung neun mögliche Positionen und einen Trigger (Abzug, Feuerknopf). Stock und Knopf sind unabhängig voneinander und für beide gibt es einen ATARI-BASIC-Befehl (soetwas fehlt bei Commodore und Sinclair). STICK(0) fragt den Steuerknüppel ab, der in Buchse (Port) 1 gestöpselt ist. STRIG(0) bezieht sich auf den entsprechenden Feuerknopf. Der gute alte ATARI 800 hatte noch vier Joystickports. Da es aber schon etwas besonderes ist, wenn bei einem Computerspiel zwei Leute miteinander spielen, wozu dann vier Anschlüsse. Kosten doch

nur Geld. Also haben die sparsamen Techniker bei den neuen XL-Modellen zwei Buchsen gestrichen.

STICK(n) und STRIG(n) finden je nach Zustand des Joysticks verschiedene Werte. STRIG(n) ist 1 wenn der Feuerknopf nicht gedrück ist und 0, wenn er gedrückt ist. Die Werte für STICK(n) entnehmen Sie bitte der Abbildung:



Halten Sie den Joystick immer so, daß der Feuerknopf oben links liegt. Ist der Steuerknüppel in Ruhestellung, findet STICK(n) den Wert 15, sonst den Zahlenwert, mit dem die entsprechende Richtung beschriftet ist.

Das folgende Demonstrationsprogramm zeigt Ihnen, wie mit diesen Befehlen gearbeitet werden kann:

```
10 S=STICK(0)
20 F=STRIG(0)
30 IF S=15 THEN ? ,,:GOTO 120
40 IF S=14 THEN ? "OBEN",,
50 IF S=13 THEN ? "UNTEN",,
60 IF S=11 THEN ? "LINKS",,
70 IF S=7 THEN ? "RECHTS",,
80 IF S=10 THEN ? "LINKS OBEN",
90 IF S=6 THEN ? "RECHTS OBEN",
100 IF S=5 THEN ? "RECHTS UNTEN",
110 IF S=9 THEN ? "LINKS UNTEN",
120 IF F=1 THEN ? "-=-"
130 IF F=0 THEN ? "BUMM"
140 GOTO 10
```

10: Der Ausdruch STICK(0) könnte im Programm wie jede andere numerische Variable verwendet werden. Da dieser ausdruck aber acht Zeichen lang ist und damit nicht nur umständlich zu tippen ist, sondern auch entsprechend viel Speicherplatz belegt, ist es praktisch, den Wert aus STICK (0) einer handlicheren Variablen zuzuweisen.

20: Das ist mit STRIG(0) nicht anders.

30: Jetzt fragen wir den Knüppel ab. Wenn S den Wert 15 enthält, ist der Joystick in Ruhe. Der PRINT-Befehl mit den zwei Kommata bewirkt, daß zwei Tabulator-Stellen nach rechts gesprungen wird.

40 bis 110: treten in Aktion, wenn der Knüppel in irgendeiner anderen Stellung steht und PRINTen einen entsprechenden Text aus.

120 und 130: fragen den Knopf ab.

140: führt zur erneuten Abfrage von STICK und STRIG zurück.

Wie schnell diese vierzehn Programmzeilen abgearbeitet werden, können Sie schon daran sehen, daß es einiges Geschick erfordert, irgendeinen Ausdruck nur ein einziges Mal auf die matte Scheibe zu bringen. Versuchen Sie's doch mal.

4.1.2 Bewegte Zeichen

Die einfachste Spielfigur, die der Computer für uns bereithält, ist ein ATASCII-Zeichen oder ein Grafikpunkt. Einem solchen Spieler wollen wir jetzt Beine machen. In GR.3+16 hat unser Spielplatz 39 mal 23 Felder. Wir setzen einen roten Grafikpunkt in die Mitte dieser Fläche. Es gibt vier Richtungen, in die er sich bewegen kann. Es liegt also nahe, für jede Bewegungsrichtung eine Taste zu belegen, z. B. L für einen Schritt nach Links, R - na Sie wissen schon. Wir können aber auch die vier Cursor-Tasten für diese Steuerung verwenden, die ja mit entsprechenden Richtungspfeilen bedruckt sind:

```
10 GRAPHICS 19
20 POKE 708,50
30 COLOR 1
40 X=19
50 Y=11
60 PLOT X,Y
70 OPEN #3,4,0,"K:"
80 GET #3,C
90 IF C=31 THEN X=X+1
110 IF C=30 THEN X=X-1
130 IF C=29 THEN Y=Y+1
150 IF C=28 THEN Y=Y-1
170 PLOT X,Y
180 GOTO 80
190 CLOSE #3
```

10: schaltet GR.3 ohne Textfenster ein.

20: wählt Farbton 50 (Rot) für COLOR 1 (Register 708).

30: ruft COLOR 1 auf.

40: setzt die Variable X auf 19. Sie wird später die Spalte bestimmen, während Y die Zeile abgibt.

50: Deshalb wird Y hier auf 11 gesetzt.

60: An die Stelle X,Y (19,11 = Mittelpunkt des Grafikfensters) wird ein Grafikpunkt gePLOTtet.

70: eröffnet den Datenkanla zur Tastatur und

80: liest (GET) eine Information, nämlich den ATASCII-Wert der gedrückten Taste, in die Variable C.

90: Die Taste CURSOR RECHTS hat den ATASCII-Wert 31. Wenn unser Grafikpunkt einen Schritt nach rechts gehen soll, so heißt das, der Spaltenwert (x) muß um 1 erhöht werden (X+1): WENN C=31 DANN X=X+1.

110: CURSOR LINKS (30) verringert den Spaltenwert (X=X-1).

130: CURSOR UNTEN (29) vergrößert den Zeilenwert (Y=Y+1).

150: CURSOR OBEN (28) verringert den Zeilenwert (Y=Y-1).

170: Nach der Veränderung des Spalten- oder Zeilenwertes, kann an die neue Stelle X,Y ein Grafikpunkt gePLOTtet werden, COLOR 1 ist immer noch aktiviert.

180: führt zu 80 zurück, damit mit dem nächsten Tastendruck der nächste Punkt gePLOTtet werden kann.

190: CLOSEt den Datenkanal, eine reine Formsache, denn dieses Programm kann Zeile 190 nie erreichen.

(Nur damit keine Mißverständnisse aufkommen: Um die Cursor-Funktion der Tasten anzusprechen, muß CONTORL gleichzeitig gedrückt werden, klar doch.)

Wenn Sie eine Taste lange genug gedrückt halten, setzt die Tastenwiederholungsfunktion ein. Sie müssen also nicht wie irre auf einer Taste herumhacken, um die gleiche Bewegung mehrmals auszulösen.

Lassen Sie das Programm einmal RUN, tasten Sie CONTROL und + (d.i. CURSOR LINKS) und halten beide Tasten gedrückt. Was geschieht: ERROR- 3 AT LINE 110

Was ist denn nun schon wieder los?

Sehen wir uns unser Machwerk einmal genauer an. Wenn wir ATASCII 30 auslösen, wird der X-Wert um 1 verringert. In Zeile 40 auf den Anfangswert 19 gesetzt, wird er 18. Das Spielchen geht weitere achtzehn Mal gut, dann ist X auf den Wert 0 heruntergekommen. Wenn wir jetzt noch einmal CURSOR LINKS drücken, setzt der gehorsame Rechner X noch einmal um 1 herunter, nämlich auf -1. Warum sollte er das auch nicht tun? Aber wenn er dann in Zeile 170 einen Grafikpunkt der Farbe COLOR 1 an die Bildschirmposition -1,11 schreiben soll, dann fragt sich die Maschine doch wohl mit Recht, wo dieser PLOT liegen soll, oder!

Als Programmierer muß man eben an alles selber denken. Und wenn sich die Gurus der schönen neuen Computer-Welt den Mund fusselig reden über "künstliche Intelligenz", keine

Maschine kann denken, sie kann immer nur ausführen, was der Programmierer vorgedacht hat. Und wenn die natürliche Intelligenz etwas vergessen hat, dann bricht die künstliche Intelligenz das Programm ab oder gibt falsche Ergebnisse aus.

Wie also können wir vorsorgen, daß durch die Zeilen 90 bis 150 keine Werte erreicht werden, die von PLOT nicht verarbeitet werden können? X darf nie größer als 39 oder kleiner als 0 werden. Diese beiden Fälle müssen wir benennen und für ihr Eintreffen Vorkehrungen treffen, indem wir befehlen, was geschehen soll, wenn X größer als 39 geworden ist.

Diese Bedingung können Sie als Programmierer ganz souverän bestimmen. Das Programm könnte z.B. abgebrochen werden, ein GOTO könnte einen ganz neuen Programmteil einleiten oder an den Anfang (Zeile 10) zurückführen.

Wenn die Bewegung des Grafikpunktes fortgesetzt werden soll, dann gibt es zwei prinzipielle Möglichkeiten. Wenn die Bewegung links aus dem Bild herausgeht (X größer 39), dann kommt sie rechts wieder herein (X=0) und umgekehrt. Oder wenn die Bewegung über den oberen Rand hinausgeht (Y kleiner 0) dann soll sie dort stehenbleiben (Y=0) und entsprechend für die anderen Ränder. Im folgenden Listing sind beide Möglichkeiten auf das vorhandene Programm angewandt. Sie hatten ja sicher schon bemerkt, daß ich entsprechende Zeilennummern ausgelassen hatte:

```
10 GRAPHICS 19
20 POKE 708,50
30 COLOR 1
40 X=19
50 Y=11
60 PLOT X,Y
70 OPEN #3,4,0,"K:"
80 GET #3,C
90 IF C=31 THEN X=X+1
100 IF X>39 THEN X=0
110 IF C=30 THEN X=X-1
120 IF X<0 THEN X=39
130 IF C=29 THEN Y=Y+1
140 IF Y>23 THEN Y=23
150 IF C=28 THEN Y=Y-1
160 IF Y<0 THEN Y=0
170 PLOT X,Y
180 GOTO 80
190 CLOSE #3
```

Fall Sie sich noch (wahrscheinlich ungern) an Herrn Boole und seine Operatoren erinnern (AND OR NOT), dann dann werden Sie nicht widersprechen wollen, daß sich diese umständlichen acht Zeilen auch kürzer fassen lassen, allerdings nur, wenn die Bewegung verharren soll, bevor sie in verbotene Zonen eintritt:

```
100 X=X+((X<39) AND (C=31))
110 X=X-((X>0) AND (C=30))
120 Y=Y+((Y<23) AND (C=29))
130 Y=Y-((Y>0) AND (C=28))
```

(und die Zeilen 90, 140, 150 und 160 bitte löschen)

100: Zu dem Wert X (Spalte) wird der Wert in der Klammer addiert. Der Wert der Klammer ist 1, wenn beide Bedingungen (AND) erfüllt sind, also wenn CURSOR RECHTS (C=31) gedrückt wurde UND noch ein Schritt nach rechts erlaubt ist (X kleiner 39). Trifft nur eine Bedingung nicht zu, ist der Wert der Klammer 0 und zu X wird 0 hinzugezählt, wovon es nicht viel hat.

110 bis 130: verfahren nicht anders.

Wenn wir nun mit dem roten Grafikpunkt über den Bildschirm spazieren, dann ziehen wir einen Rattenschwanz von Grafikpunkten hinter uns her. Soll sich nur ein einzelner Punkt bewegen, dann müssen wir die alte Position löschen, wenn wir die neue drucken:

```
85 COLOR 0:PLOT X,Y
165 COLOR 1
```

85: bevor die Koordinaten entsprechend dem Tastendruck (C) verändert werden, löscht Farbe 0 (Hintergrund) den alten Punkt.

165: Bevor der neue Punkt gePLOTtet wird, müssen wir nun erst Farbe 1 einschalten, noch ist ja COLOR 0 aufgerufen. In der folgenden Zeile wird dann der Punkt gedruckt. Falls die Koordinaten inzwischen nicht verändert wurden, wird der Punkt eben wieder an die alte Stelle gesetzt. Allerdings führt diese Vorgehensweise zu einem leichten Flimmern durch das Ein- und Ausschalten.

Wenn Sie diese beiden Zeilen in das bestehende Programm einfügen, streunt nur noch ein einsames Quadrat durch die Finsternis des leeren Bildschirms.

Wenn wir uns als nächstes dem Spaß mit dem Joystick zuwenden, dann ist nicht mehr viel Neues zu lernen. Ob Stock oder Taste, der Programmaufbau ist fast gleich:

```
10 REM JOYMOVE1.BIR
20 ? CHR$(125)
30 POKE 752,1
40 SPA=19
50 ZEI=11
60 ZCH=0
70 ZCH=ZCH+((STRIG(0)=0) AND (ZCH<254))
80 IF ZCH>255 THEN ZCH=0
90 SPA=SPA+((STICK(0)=7) AND (SPA<39))
100 SPA=SPA-((STICK(0)=11) AND (SPA>0))
110 ZEI=ZEI+((STICK(0)=13) AND (ZEI<23))
120 ZEI=ZEI-((STICK(0)=14) AND (ZEI>0))
130 POSITION SPA,ZEI
140 ? CHR$(ZCH)
150 GOTO 70
```

10: Ein REM-Denkzettel

20: ?? Was wird denn da geprintet? Ja hab ich denn nach 166 Seiten noch nicht verraten, daß die Editierkommandos als PRINT-Anweisung programmiert werden können? Es gibt dafür zwei Möglichkeiten. Entweder der ATASCII-Code wird als CHR$(n) gePRINTet oder das Zeichen selbst als String. Nun werden aber die Editierkommandos durch Grafikzeichen dargestellt, die ein normaler Drucker nicht ohne weiteres aufs Papier zu bringen vermag. Deshalb bevorzuge ich hier den CHR$-Befehl. 125 ist der Code für CLEAR (Bildsschirmlöschen). Sie können Zeile 20 auch so schreiben ?"(Taste ESC)(Taste CLEAR)", dann haben Sie einen kleinen geknickten Pfeil als String und der löscht den Bildschirm. Entsprechen können Cursorbewegungen, Löschen und Einschieben von Zeichen und Zeilen und TAB-Kommandos in ein Programm eingebaut werden. Die Taste ESC (ESCape = entkommen) wirkt immer nur für einen einzigen direkt folgenden Tastendruck.

30: schaltet den Cursor aus.

40: Die Variable SPA wird auf 19 gesetzt. Sie wird später die Spalte bestimmen.

50: Die Variable ZEI, hier auf 11 gesetzt, bestimmt später die Zeile.

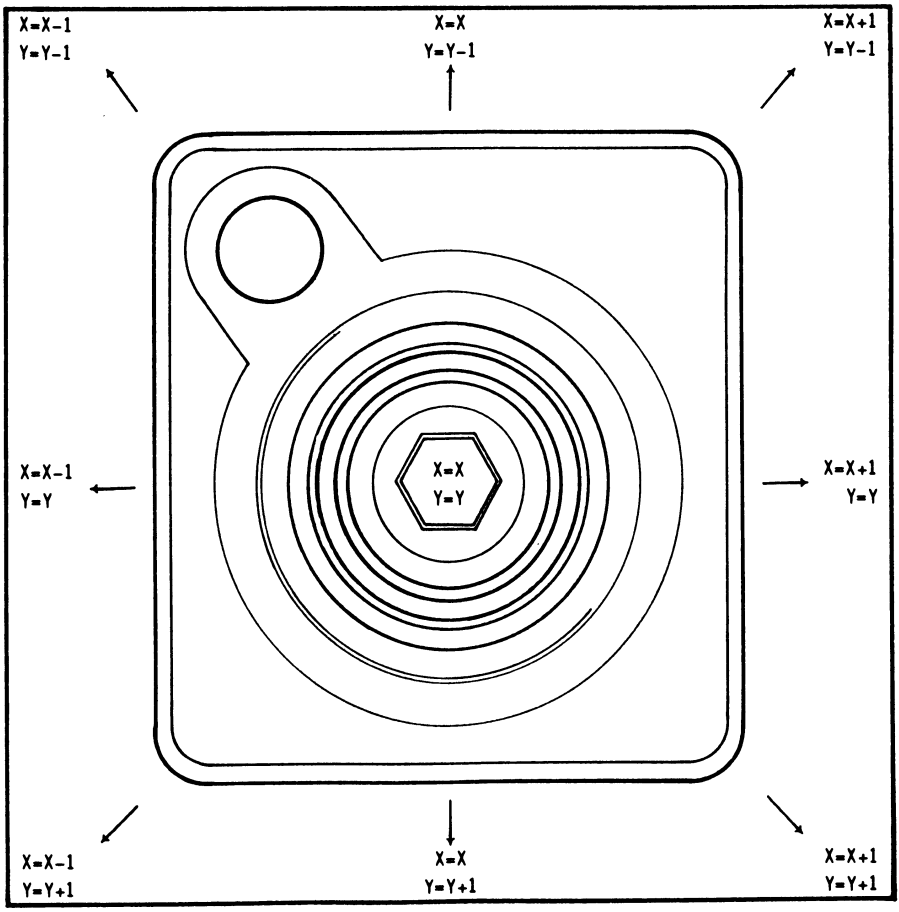
60: Die Variable ZCH wird bestimmen, welches Zeichen auf den Bildschirm gedruckt wird. Ersteinmal bekommt sie den Wert 0.

70: Schon wieder der alte Boole (spricht sich übrigens 'bu:l')! Wenn der Feuerknopf gedrückt ist (STRIG(0)=0) AND

die Variable ZCH kleiner als 254 ist, dann ist der Boolesche Ausdruck in der Klammer wahr, hat also den Wert 1, und ZCH wird um 1 erhöht (ZCH=ZCH+1). Ist der Ausdruck unwahr, die Klammer gleich 0, wird ZCH nicht erhöht. Ist doch ganz einfach.

80: Wenn ZCH größer als 255 ist, dann wird ZCH hier auf 0 gesetzt - weil der Zeichensatz nur 256 (0 bis 255) Zeichen umfaßt.

Sie habens natürlich gemerkt. Zeile 80 ist wirkungslos, weil durch die Bedingung in Zeile 70 der Wert 255 gar nicht erreicht werden kann. Fragen Sie in Zeile 70 den Feuerknopf mit IF-THEN ab oder lassen Sie Zeile 80 fort. Die beiden Varianten führen zu unterschiedlichen Ergebnissen. Im ersten



Fall kann kein neues Zeichen mehr mit dem Feuerknopf aufgerufen werden, wenn einmal der Wert 255 erreicht ist; im zweiten Fall gehts dann wieder mit Zeichen 0 von vorne los.

90 bis 120: verändern die SPAlte oder die ZElle je nach STICK(0)-Eingabe.

130: POS.itioniert den unsichtbaren Cursor auf SPA,ZEl und

140: wandelt den numerischen Wert ZCH durch CHR$ in das entsprechende ATASCII-Zeichen um und ?et es auf den Bildschirm.

Mit dem Joystick können Sie also auf dem Bildschirm hin und her steuern und durch Druck auf den Trigger das Zeichen wechseln. Damit das nicht nur einmal klappt, führt

150: nach 70 zurück.

Probieren Sie's mal aus. (RUN!)

Wenn Sie eine Weile auf der Mattscheibe Buchstabensuppe verkleckert haben, dann machen Sie bald ein paar überraschende Erfahrungen. Probieren Sie folgendes. Sie fahren von der Mitte bis zum linken Rand. Wechseln Sie öfter mal das Zeichen. Jetzt gehts nach oben, alles ruhig, dann nach rechts bis zum Anschlag, jetzt runter. Halt! Nur ein paar Zeilen. Gehen Sie jetzt ein paar Schritte nach rechts, dann ein gutes Stück nach unten und zurück zum rechten Rand. So und jetzt tippen Sie nur ganz sacht gegen den Knüppel, damit Sie nur einen einzigen Schritt nach oben machen. Was geschieht?! Und warum!

Jetzt fahren Sie mal zum unteren Rand. Ab geht die Fahrt! Wenn Sie jetzt mit dem Joystick nur noch nach rechts und links steuern, können Sie einen wilden Slalom fahren und dabei ATASCII-Zeichen auf der Strecke lassen. Warum?

Verantwortlich hierfür ist die Bildschirmverwaltung von GRAPHICS 0. Wird ein Zeichen in Spalte 39 gedruckt, springt der Cursor (auch wenn er unsichtbar gePOKEt wurde) sofort an den Anfang der nächsten Zeile und alle folgenden Zeilen werden um eins nach unten verschoben. Wenn Zeile 23 verlassen wird, müßte der Cursor nach Zeile 24 springen. Weil das nun mal nicht geht, werden alle Zeilen darüber um eins nach oben geschoben. Die oberste Zeile verschwindet dabei aus dem Bildschirm.

4.1.3 Genossen und Geschosse

Mit Grafikpunkten oder ATASCII-Zeichen läßt sich eine Menge auf dem Bildschirm bewegen, aber Spielfigur und Spielfeld sind aus dem selben Holz geschnitzt, ihre Daten belegen den gleichen Speicher. Die Bewegungen des Spielers verändern das Spielfeld.

Deshalb wurde die Sprite-Idee entwickelt. Das ist kein Erfrischungsgetränk sondern der Name für ein Zeichen, das in Farbe, Form und Bewegung autonom von der sonstigen Grafik definiert werden kann. Ein Sprite liegt gleichsam wie eine Transparentfolie über dem Grafikfenster.

Das Programmieren von Sprites ist bei jedem Heim-Computer mehr oder weniger kompliziert gelöst, meist mehr. Bei ATARI heißt das Kunststück "Player-Missile-Graphic" (Spieler-Geschoß-Grafik) und ist nicht ein Kapitel für sich sondern ein ganzes Buch. Deshalb hier nur eine Kostprobe vom großen Kuchen.

Um PM-Grafik zu programmieren, muß man einiges über die Speicherbelegung wissen und etliche Befehle durch direkten Speicherzugriff erteilen. Aber am Ende dieses Kapitels werden Sie zumindest einen Player gestalten und auf dem Bildschirm bewegen können.

Insgesamt können vier Player (0 bis 3) und vier Missiles (0 bis 3) unabhängig von einander definiert werden. Die vier Missiles können auch zu einem fünften Player zusammengefaßt werden. Die Adressen der Player 0 bis 3 sind 53256 bis 53259. In diese Adressen wird durch eine Kennzahl die Breite des Players auf dem Bildschirm festgelegt. Die Werte 0 oder 2 ergeben eine normale Breite (16 Bildpunkte), 1 die doppelte und 3 die vierfache Breite. Also POKE 53256,0 bestimmt für Player 0 die normale Breite. Das muß allerdings nicht extra gePOKEt werden, weil 0 der Grundzustand der Adresse ist.

Mit POKE 53277,3 wird PM-Grafik eingeschaltet. Als nächstes muß in Adresse 54279 die Speicheradresse abgelegt werden, bei der die Daten der PM-Grafik beginnen.

Wir schauen unter Adresse 106 nach, bis wohin der Speicher

bereits belegt ist. Der Wert in Adresse 106 gibt den freien Speicherplatz in pages (Seiten) an. Ein page faßt 256 Byte. PEEK(106) liefert den gesuchten Wert. Indem wir davon 8 abziehen, reservieren wir acht Seiten im Speicher für unsere PM-Grafik. Mit diesem Wert werden Sie meist zurechtkommen. Diesen Wert müssen wir also in die genannte Adresse schreiben: POKE 53279,PEEK(106)-8

Die Startadresse, bei der beginnend der Speicher mit den Daten unserer PM-Tabelle beschrieben werden kann ist also: TABSTART=256*PEEK(106)-8.

Die Daten für Player und Missiles müssen in einer vorgegebenen Reihenfolge in den Speicher geschrieben werden, das nennt man Speicherorganisation. Wie die acht pages (=2048 Byte) aufgeteilt sind, zeigt Ihnen die Tabelle unten. Die Zahlen für den Speicherbereich beziehen sich jeweils auf die Startadresse, die ich TABSTART genannt habe. Die Daten für Player 0 bei doppelzeiliger Auflösung beginnen also 512 Speicherzellen tiefer als TABSTART und belegen die folgenden 128 Adressen, jede Adresse faßt ein Byte. Für jeden Player sind also 128 Byte Speicherplatz reserviert. Diese Daten bestimmen seine grafische Form. Aus jedem einzelnen Bit (ein Byte setzt sich bekanntnlich aus acht Bit zusammen) wird ein PM-Grafikpunkt errechnet. Jeder dieser Grafikpunkte ist bei normaler Breite zwei Bildpunkte breit und bei doppelzeiliger Auflösung zwei Bildschirmzeilen hoch. Bei doppelter Breite ist ein PM-Grafikpunkt vier Bildpunkte breit; bei einzeiliger Auflösung nur eine Bildschirmzeile hoch.

<table>
<tr><th colspan="4">doppelzeilig</th><th colspan="4">einzeilig</th><th>Speicherbereich</th></tr>
<tr><td>M3</td><td>M2</td><td>M1</td><td>M0</td><td colspan="4"></td><td>384 - 511</td></tr>
<tr><td colspan="4">Player 0</td><td colspan="4"></td><td>512 - 639</td></tr>
<tr><td colspan="4">Player 1</td><td colspan="4"></td><td>640 - 767</td></tr>
<tr><td colspan="4">Player 2</td><td>M3</td><td>M2</td><td>M1</td><td>M0</td><td>768 - 895</td></tr>
<tr><td colspan="4">Player 3</td><td>M3</td><td>M2</td><td>M1</td><td>M0</td><td>896 - 1023</td></tr>
<tr><td colspan="4" rowspan="4"></td><td colspan="4">Player 0</td><td>1024 - 1279</td></tr>
<tr><td colspan="4">Player 1</td><td>1280 - 1535</td></tr>
<tr><td colspan="4">Player 2</td><td>1536 - 1791</td></tr>
<tr><td colspan="4">Player 3</td><td>1792 - 2048</td></tr>
</table>

Bit	7	6	5	4	3	2	1	0
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Dezimalwert	128	64	32	16	8	4	2	1

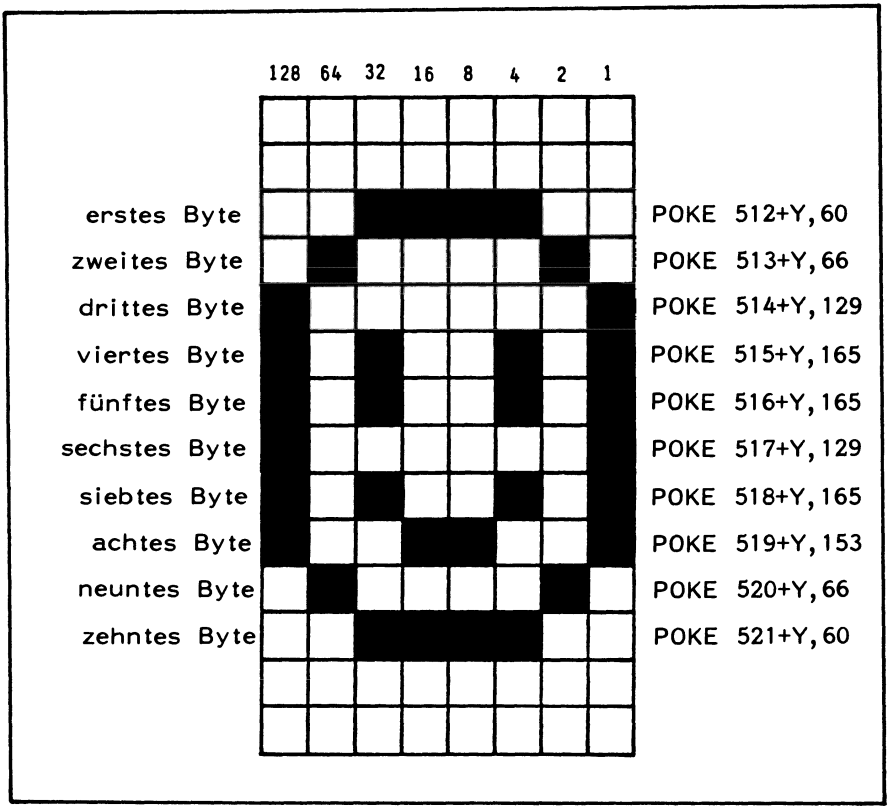
Jedes der acht Bit eines Speicherbyte bestimmt einen Grafikpunkt. Ist das Bit gesetzt, wird ein Grafikpunkt ausgegeben, ist das Bit nicht gesetzt, wird kein Grafikpunkt ausgegeben.

Ein Bit kennt nur den Zustand EIN oder AUS, der als 0 oder 1 dargestellt wird. Innerhalb des Byte ist jedem Bit eine Zweierpotenz als Dezimalwert zugeordnet. Das niederwertige Bit 0 entspricht 2 hoch 0 (=1), kann also gesetzt 1 und nicht gesetzt 0 abgeben. Das höchstwertige Bit 7 entspricht dem Dezimalwert 128, wenn es gesetzt ist. Das Byte als ganzes kann also Dezimalwerte von 0 (alle Bits nicht gesetzt) bis 255 (alle Bits gesetzt) annehmen.

Bildlich gesprochen malen wir die Grafikpunkte unseres Players Bit für Bit in ein Byte und addieren die Dezimalwerte der gesetzten Bits. So erhalten wir einen Dezimalwert, der als Wert in eine Speicherzelle des Player-Speicherbereiches geschrieben werden muß.

Jedes Byte umfaßt acht PM-Grafikpunkte, die in einer Zeile nebeneinander liegen. Zeile für Zeile wird der Player zusammengesetzt, die Bit-Muster der einzelnen Bytes ergeben seine Form. Da die Speicherorganisation 128 Adressen für einen Player freihält, kann er 128 Grafikpunkte hoch sein.

Bit	7	6	5	4	3	2	1	0	
	■	□	■	□	□	■	□	■	= 165
Dezimalwert	128	64	32	16	8	4	2	1	



Ein Player ist also ein Byte breit und das entspricht je nach Auflösung 16, 32 oder 64 Bildpunkten und 128 Byte hoch, was bei doppelzeiliger Auflösung 256 Bildschirmzeilen entspricht. Bei einzeiliger Auflösung ist der Player 256 Bytes hoch, also auch 256 Bildschirmzeilen. Die Player sind also höher als das Grafikfenster (192 Bildschirmzeilen), das heißt aber nicht, daß auch alle diese Zeilen für den Player gestaltet werden müssen.

Die Abbildung oben zeigt Ihnen das Bit-Muster des Players, den Sie in dem folgenden Programmbeispiel wiederfinden werden. Er ist zehn Bytes hoch. Rechts neben dem Bit-Muster sehen Sie die entsprechende POKE-Anweisung, beginnen mit dem ersten Speicherplatz für Player 0 (512), in den der Dezimalwert (60) für das entsprechende Bitmuster geschrieben wird. Der Wert Y bestimmt die vertikale Position des Players und der Wert der Startadresse muß noch addiert werden.

			Instruction Fetch DMA	einzeilig doppelzeilig	Player DMA	Missile DMA		ANTIC DMA	
	Bit	7	6	5	4	3	2	1	0	
SDMCTL	Register 559	0	0	1	0	1	0	0	1	= 42
	Dezimalwert	128	64	32	16	8	4	2	1	

Eine besondere Bedeutung kommt dem Register 559 (Kurzbezeichnung SDMCTL) zu. Stellen Sie sich das Register wie eine Reihe von Schaltern vor. Jedes einzelne Bit ist ein Schalter, mit dem bestimmte Wirkungen eingeschaltet werden können.

Normalerweise steht in 559 eine 34. Dieser Wert entsteht, weil Bit 5 (Dezimalwert 32) und Bit 1 (=2) gesetzt sind. Dadurch erfährt ANTIC, daß er ein Grafikfenster von 320 Bildpunkten Breite und 192 Bildschirmzeilen Höhe ausgeben soll. Werden durch POKE 559,33 Bit 5 und Bit 1 gesetzt, verändert sich die Form des Grafikfensters. Am besten können Sie das beobachten, wenn Sie in GR.0 ein Listing oder sonst irgendeinen Text auf dem Bildschirm haben und dann POKE 559,33 als Direkteingabe vornehmen.

Eine weitere Variante ist möglich, wenn Bit 5, 1 und 0 (=35) gesetzt werden. Um das Durcheinander auf der Mattscheibe wieder in die gewohnte Ordnung zu bringen, geben Sie POKE 559,34 direkt ein.

Für PM-Grafik müssen weitere Bit-Schalter betätigt werden. Bit 4 bestimmt die Zeilenauflösung. Wird hier eingeschaltet, ist die Auflösung einzeilig und die Player-Daten müssen ab 1024 der Speicherorganisation abgelegt werden; steht hier eine 0, ist die Auflösug doppelzeilig und die Playerdaten beginnen bei 512.

Sollen nur Player definiert werden, muß Bit 3 (=8) gesetzt werden. Sollen auch Missiles zum Einsatz kommen, muß auch Bit 2 (=4) gesetzt werden.

Das Register 559 steuert den Direct Memory Access (direkter Speicherzugriff). Normalerweise sind Bit 5 und 1 (=32) ge-

setzt. Wenn Sie auch diese beiden Bits ausschalten (POKE 559,0), erfolgt keine Ausgabe an den Bildschirm. Dadurch wird sehr viel Rechnerzeit gespart. Ein Programm kann so fast ein Drittel schneller laufen. Wenn die Bildschirmausgabe wieder benötigt wird, dann schaltet POKE 559,34 den Grundzustand wieder ein.

Die horizontale Position des Players auf dem Bildschirm wird durch einen entsprechenden Wert im vorgesehenen Register bestimmt. Die Adressen für die horizontale Position von Player 0 bis 3 sind 53248 bis 53251, für Missile 0 bis 3 53252 bis 53255. Der Wert 0 bestimmt die Positon ganz links, der Wert 255 ganz rechts.

Die vertikale Position des Players wird dadurch bestimmt, wo sich in seinem Speicherbereich die Daten befinden. Der Player reicht über die gesamte Höhe des Bildschirms (und noch ein Stückchen weiter). Seine Gestalt wird durch Bit-Muster bestimmt. Üblicherweise ist der Player nur einige Bytes hoch. Werden diese Daten in einen anderen Speicherbereich des Players verlegt, so erzeugt das Bit-Muster ein Bild an einer vertikal verschobenen Position.

Um den Player hin und her zu bewegen, muß also nur ein Wert in einer Adresse geändert werden. Das geht fix. Um den Player aber rauf und runter zu bewegen, müssen seine gesamten Daten aus den alten Adressen in die neuen geschaufelt werden. Das braucht seine Zeit.

Bevor ich nun endlich das Programmbeispiel auf Sie zulasse, müssen wir uns noch kurz über die Farbe unterhalten. Die Farbregister 704 bis 707 (die sonst nur bei GR.10 genutzt werden) nehmen die Farbwerte für Player+Missile 0 bis 3 auf. Und in der Adresse 623 wird über die Prioritäten entschieden. Steht hier eine 1, befinden sich alle Player vor der normalen Grafik. Mit einer 4 befindet sich die Grafik vor allen Playern. Mit einer 2 erscheinen Player 0 und 1 vor der Grafik, Player 2 und 3 dahinter. Mit einer 8 sind Grafikpunkte, die sich auf Farbregister 708 oder 709 beziehen vor den Playern, die Grafikpunkte in Farben aus Register 710 und 711 dahinter.

Falls Ihnen jetzt doch ein wenig der Kopf schwirren sollte, dann tippen Sie einfach mal das folgende Programm ein, stöpseln den Joystick in Port 1 und jagen das grinsende Gesicht hinter den rosa Gitterstäben hin und her. Danach allerdings sollten wir das Player-Missile-Programm doch noch einmal zusammen Zeile für Zeile durchgehen, oder?:

```
100 GRAPHICS 19
110 POKE 708,50:POKE 710,54
120 COLOR 3
130 FOR Y=0 TO 23
140 FOR X=0 TO 39
150 PLOT X,Y
160 NEXT X
170 NEXT Y
180 COLOR 1
190 FOR X=2 TO 39 STEP 5
200 PLOT X,0:DRAWTO X,23
210 NEXT X
220 FOR Y=4 TO 23 STEP 8
230 PLOT 0,Y:DRAWTO 39,Y
240 NEXT Y
1000 REM PM-GRAFIK
1010 POKE 623,8
1020 X=120:Y=60
1030 P=PEEK(106)-8
1040 POKE 54279,P
1050 TABSTART=P*256
1060 POKE 704,10
1070 POKE 559,42
1080 POKE 53277,3
1090 POKE 53256,0
1100 POKE 53248,X
1110 FOR J=TABSTART+512+Y TO TABSTART+521+Y
1120 READ D
1130 POKE J,D
1140 NEXT J
1150 DATA 60,  66,129,165,165,129,165,153,66,60
1200 REM BEWEGEN DES PLAYERS
1210 M=STICK(0)
1220 IF M=15 THEN 1210
1230 IF M=7 THEN X=X+1
1240 IF M=11 THEN X=X-1
1250 POKE 53248,X
1260 IF M=13 THEN GOSUB 2000
1270 IF M=14 THEN GOSUB 2100
1280 GOTO 1210
2000 FOR J=10 TO 0 STEP -1
2010 POKE TABSTART+512+Y+J,PEEK(TABSTART+511+Y+J)
2020 NEXT J
2030 Y=Y+1
2040 RETURN
2100 FOR J=0 TO 10
2110 POKE TABSTART+511+Y+J,PEEK(TABSTART+512+Y+J)
```

```
2120 NEXT J
2130 Y=Y-1
2140 RETURN
```

100: Der PM-Grafik ist es völlig egal, in welchem Grafikmodus mit oder ohne Textfenster der Hintergrund aufgebaut wird.

110 bis 240: bauen diesen Hintergrund oder Spielfeld auf. Es besteht aus Grafikpunkten, die vom Farbregister 710 mit COLOR 3 gefärbt werden. Diese Grafikpunkte werden später hinter dem Player liegen. Dann werden mit COLOR 1 Gitterstäbe gezogen. Und die sollen natürlich vor dem Player liegen.

1000: Hier beginnt die PM-Grafik. Die Reihenfolge der nächsten zehn Zeilen ist unbedeutend.

1010: bestimmt die Prioriät. Der Grinse-Player befindet sich hinter den Gitterstäben. Es wäre aber denkbar, durch Druck auf den Feuerknopf der Wert in dieser Adresse auf 1 zu verändern. Probieren Sie aus, was dann geschieht, indem Sie in dieser Zeile mal eine 1 POKEn.

1020: Werte für die horizontale (X) und vertikale (Y) Lage des Players werden in Variablen notiert.

1030: Hier werden acht Seiten Speicherplatz reserviert. Da dieses Programm nur mit doppelzeiliger Auflösung arbeitet, würden vier Seiten ausreichen.

1040: Der in 1030 ermittelte Wert wird in Adresse 54279 abgelegt, und

1050: in der Variablen TABSTART (die auch SONSTWIE heißen könnte) wird die Anfangsadresse für die PM-Tabelle abgelegt.

1060: Für Player 0 wird in Register 704 die Farbe 10 (fast Weiß) bestimmt.

1070: Instruction Fetch DMA, Player DMA und normales Spielfeld werden eingeschaltet.

1080: schaltet PM-Grafik ein. Probieren Sie hier ruhig einmal andere Werte aus. Durch falsches Programmieren kann ja nichts am Computer beschädigt werden. Aber vielleicht machen Sie ein paar interessante Entdeckungen mit den Werten 1 oder 2.

1100: Hier wird die horizontale Position für Player 0 abgelegt und durch Veränderung dieses Wertes kann der Spieler später hin- und hergeschoben werden.

1110: Jetzt werden die Daten für den Player in die entsprechenden Speicheradressen eingelesen. Wie der Player aussehen soll und welche Dezimalwerte ihn darstellen, wurde in der Grafik auf Seite 173 ermittelt. Er ist zehn Datenbytes hoch, es müssen also zehn Adressen beschrieben werden. Die Speicherorganisation sieht vor, daß der doppelzeilig aufgelöste Player 0 von 512 bis 639 reicht (s. Tabelle Seite 171). Der Spieler soll aber nicht in der obersten Adresse beginnen (oben im Bildschirm liegen), sondern um Y (=60) Zeilen tiefer. Wir schreiben die Datenbytes also ab 512+Y in die Speicher. In der Variablen TABSTART haben wir die Anfangsadresse der Player-Missile-Tabelle notiert. Diesen Wert müssen wir nun noch addieren, dann haben wir die Nummer der Speicherzelle, in die das erste Datenbyte unseres Players geschrieben werden muß. Das letzte Byte wird zehn Adressen tiefer geschrieben, die zehn Bytes müssen also von TABSTART+512+Y bis TABSTART+521+Y abgelegt werden. Und das besorgen wir durch eine FOR-NEXT-Schleife, deren Schleifenzähler J in dieser Zeile von der ersteren bis zur letzteren Adresse definiert wird.

1120: liest die Dezimalwerte für den Player in die Variable D, die DATA-Zeile folgt unten.

1130: In die vom Schleifenzähler J bestimmte Adresse wird der jeweils gelesene Dezimalwert D gePOKEt.

1140: beendet die Schleife.

1150: enthält die DATA des Players als Dezimalwerte. Wenn Sie hier ein RUN geben, können Sie den Player schon mal anschauen,

1200: aber er soll ja auch noch bewegt werden.

1210: Dazu fragen wir STICK(0) ab.

1220: Wenn der Stick in Ruhestellung ist, wird sofort zur neuerlichen Abfrage zurückgesprungen.

1230: Wird der Steuerknüppel nach rechts gedrückt, soll sich der Player natürlich nach rechts bewegen, seine horizontale Position wird vergrößert.

1240: Bei der Bewegung nach links verkleinert sich der Wert für die horizontale Position.

1250: Hier wird der veränderte (oder alte) Wert der horizontalen Position in die entsprechende Adresse gePOKEt.

1260: Soll sich der Player vertikal bewegen, dann sieht die Sache etwas anders aus. Die Werte des Players müssen aus

den alten Adressen gelesen und in die neuen Adressen hineingeschrieben werden, die der neuen Position entsprechen. Für die Bewegung nach unten wird hier die Subroutine von Zeile 2000 angesprungen. Das ist nicht notwendig, es erlaubt nur eine bessere Übersichtlichkeit, weil dann jedes Kommando in einer gesonderten Zeile stehen kann. Da die BASIC-Zeile eine solche Portion verkraftet, können Sie aber auch den Inhalt der Zeilen 2000 bis 2030 statt des GOSUB 2000 in dieser einen Zeile (durch : getrennt) unterbringen.

1270: Sprung zur Subroutine für Bewegung nach oben.

1280: Rücksprung zur erneuten Joystick-Abfrage.

2000: Subroutine für die Bewegung nach unten. Der Inhalt der zehn Register muß gelesen und dann je eine Adresse tiefer abgelegt werden. Um die Adresseninhalte nicht zu überschreiben, beginnen wir mit der tiefsten Adresse TABSTART+512+Y+10. Diese Adresse ist leer (=0), denn die Daten des Players hören eine Adresse höher auf. In dieser Adresse TABSTART+512+Y+9 lesen wir den Wert (PEEK) und POKEn ihn in die oben genannte Adresse.
Die FOR-NEXT-Schleife geht die Werte einzeln rückwärts durch, bis als letztes in die Adresse TABSTART+512+Y+0 der Wert aus der nächsthöheren Adresse gePOKEt wird, und der ist 0. Würden wir die jeweils letzte Speicherzelle im elften Arbeitsgang nicht löschen, würde der Player Linien hinter sich herziehen.
Wie das gemeint ist, können Sie ausprobieren: Steuern Sie mit dem Player immer weiter nach oben. Er verschwindet aus dem Bild, d.h. da wir keine entsprechende Sicherung eingebaut haben, werden Speicherzellen abgefragt, die für unseren Player 0 gar nicht mehr zuständig sind. Wenn Sie dann wieder nach unten fahren, kann es sein, daß durch den Vorgang des Umschreibens der Speicherinhalte in die jeweils tiefere Adresse ein fremder Speicherinhalt verschleppt wird, der, wenn er den Speicherbereich des Players erreicht, in Grafikpunkte umgewandelt wird. Die letzte Adresse enthält dann keine 0 mehr, die hinter dem Player löscht, sondern irgendeinen Wert, der in irgendwelche Grafikpunkte umgesetzt Zeile um Zeile ausgegeben wie eine Schleifspur auf dem Weg nach unten hinterhergezogen wird.

2010: Hier wird in die tiefere Adresse der Wert aus der nächst höheren geschrieben.

2020: Der Vorgang wird 11mal (10 bis 0) wiederholt.

2030: Um auf dem Laufenden zu bleiben, muß abschließend der Wert für die vertikale Position (Y) noch um 1 vergrößert

werden (= Bewegung nach unten).

2040: beendet die Subroutine vorschriftsmäßig mit RETURN.

2100: Wenn die Speicherwerte nach oben verlegt werden sollen, dann gehen wir genau umgekehrt vor.

2110: Wir PEEKen die höchste Adresse und POKEn ihren Wert noch eine Speicherzelle höher.

2120: Das tun wir ebenfalls 11mal, damit der letzte Wert gelöscht wird.

2130: Dann verringern wir den Wert der vertikalen Position

2140: und beschließen die Subroutine.

Natürlich ist es auch möglich, den Player diagonal zu bewegen. Auf den entsprechenden Joystick-Wert werden sowohl die horizontale als auch die vertikale Bewegung ausgeführt. (Dazu müßten natürlich noch ein paar Programmzeilen eingefügt werden, muß ich das extra erwähnen?)

Wenn Sie jetzt mit dem Grinse-Player herumfahren, sehen sie, wie schnell er nach rechts und links gesteuert werden kann und wie lange das Umschreiben der Speicherinhalte dauert. (Und Sie ahnen auch, warum bei manchen Computer-Spielen nur Rechts-links-Bewegungen möglich sind.)

Der Player ist von Grafikfenster unabhängig. Sie können mit ihm Bildschirmbereiche besuchen, die außerhalb des Grafikfensters liegen. Wenn Sie nach rechts oder links aus dem Bild herausfahren, dann kommt irgendwann ein ERROR- 3.

Nach oben und unten können Sie ganz schön lange fahren, wenn Sie wollen, bis sie dadurch irgendwann eine Speicherzelle erreichen und ihren Wert zerstören, der für irgendeine andere Funktion wichtige Daten enthält. Es ist deshalb sinnvoll durch eine entsprechende IF-THEN-Abfrage zu verhindern, daß die Playerdaten aus dem ihnen zugewiesenen Speicherbereich hinauswandern.

Wenn Sie BREAK drücken, werden die Playerdaten nicht gelöscht, aber der Wert in 53277 wird verändert. Es entsteht an der Stelle des Players ein ungeordnetes Flimmern. Sie brauchen sich keine Sorge um Computer oder Fernseher zu machen. Beiden ist ziemlich egal, was sie ins Bild setzen. An dem Flimmern können Sie aber sehr schön die gesamte Ausdehnung des Players erkennen.

Wenn Sie das Programm jetzt erneut starten, dann haben Sie wahrscheinlich anfangs zwei gleiche Player in der selben Spalte auf dem Bildschirm, aber nur einer davon bewegt

sich auch vertikal, wenn Sie den Steuerknüppel nach oben oder unten drücken. In dem Moment, als Sie die BREAK-Taste gedrückt haben, befand sich der Player in Form von Daten in bestimmten Speicherzellen. Und diese Daten wurden durch BREAK auch nicht gelöscht. Als Sie das Programm neu gestartet haben, wurden die Playerdaten wieder in die Adressen der Anfangsposition geschrieben. Auf dem Bildschirm erscheinen dann alle diese Daten als Grafikpunkte. Der alte Player behält seine vertikale Position bei, die Daten des neuen Players wechseln die Register. Durch seine Bewegung kann der neue Player den alten erreichen und löschen. Um aber diese Datenverdoppelung zu vermeiden ist es sinnvoll, bei Programmbeginn den gesamten Speicherbereich des Players zu löschen:

```
FOR J=TABSTART+512 TO TABSTART+639:POKE J,0 (=128 Adr.)
```

Abschließend für Tüftler und Jung-Hacker weitere Tips in Kurzform:

Missiles werden wie Player behandelt. Ihre Bewegung sollte nur einmal initiiert selbsttätig ablaufen. Durch Dosierung der X- und Y-Werte können Geschosse in jede Richtung fliegen. Ein Missile ist in der normalen Auflösung nur ein Bit (= 2 Bildpunkte) breit. Die horizontale Position für Missile 0 bis 3 liegt in Adressen 53252 bis 53255. Missiles 0 bis 3 können auch zu Player 4 zusammengefaßt werden.

Geschosse müssen auch treffen. Kollisionsregister:

Missile-Spielfeld 0 bis 3	in	53248 bis 53251
Player-Spielfeld 0 bis 3	in	53252 bis 53255
Missile-Player 0 bis 3	in	53256 bis 53259
Player-Player 0 bis 3	in	53260 bis 53263

Und wenn es gekracht hat, muß das Kollisionsregister 53278 durch irgendeinen Wert ungleich 0 gelöscht werden, damit es für den nächsten Zusammenstoß vorbereitet ist.

Was sich hinter diesen geheimnisvollen Registern verbirgt (und noch wichtiger: in ihnen), das kann an dieser Stelle nicht mehr im Detail erklärt werden, sonst würde das Buch aus den Deckeln platzen. Wenn Sie aber Spaß an Denksportaufgaben haben, dann können Sie mit Probieren, Beobachten, Nachdenken und Vergleichen sich selbst Ihren PEEK und POKE darauf machen.

4.2 Das Spielfeld

Das Spielfeld ist der begrenzte Platz, auf dem das Spiel stattfindet. Das Spielfeld ist sichtbarer Ausdruck der Spielregeln. Seiner Gestaltung kommt deshalb große Bedeutung zu.

Bei Computer-Spielen ist der TV-Schirm der Schauplatz des Geschehens. In der einfachsten Form kann die Mattscheibe wie eine Anzeigetafel verwendet werden und als Kommunikationsmedium zwischen Spielleiter (Computer) und Spieler fungieren. Die Ausgabe erfolgt in Zeichen, aber selbst wenn mit grafischen Elementen oder richtiger Grafik immer mehr gestaltet wird, bleibt doch das Prinzip der Schautafel erhalten.

Eine Übergangsform ist, wenn die Bildschirmgrafik nur einen Hintergrund abgibt, vergleichbar mit einer Kulisse, vor der das Spiel abläuft, die aber auf das Spielgeschehen nicht einwirkt. Bei dem Player-Missile-Programm im vorherigen Kapitel hatte die Grafik eine solche Funktion.

Interessanter wird es jedoch, wenn der Bildschirm zum Parcours wird, wenn also bestimmte Teile der Grafik wechselnde Wirkungen haben, die Geschwindigkeit des Spielers verlangsamen (Morast), undurchdringlich sind (Dornenhecke) oder gar tödlich wirken (Fallgrube).

Es gibt zwei grundlegende Möglichkeiten, ein solches "aktives" Spielfeld zu gestalten. Die direktere Methode besteht darin, verschieden wirksame Teile des Spielfelds in verschiedenen Farben zu PLOTten. Mit dem LOCATE-Befehl können die COLOR-Werte im Spielverlauf gelesen und entsprechend verarbeitet werden. Das bedingt jedoch, daß es nur so viele Spielfeldwirkungen gibt, wie Farbwerte zur Verfügung stehen und das Spielfeld keine Geheimnisse enthält. Der Bildschirmspeicher ist gleichzeitig der Schiedsrichter, der das Spielgeschehen steuernd überwacht.

Die zweite Methode ist etwas aufwendiger, dafür aber auch vielseitiger. Die Schiedsrichter-Informationen werden unabhängig von der Grafik und ihrem Bildschirmspeicher aufbewahrt. Dazu bietet sich eine doppelt indizierte Variable an, die so viele "Fächer" wie das Grafikfenster Positionen

hat. Jede Bildschirmposition ist dann der entsprechenden Variablen zugeordnet, und die Variable kann beliebige Werte für beliebige Reaktionen beinhalten, die auf dem zugeordneten Spielfeldplatz wirksam werden. Dieser doppelt indizierte Schiedsrichter ist für den Spieler unsichtbar.

Zuerst also ein Beispiel für das LOCATE-Prinzip. Das Spielfeld besteht aus einem breiten Burggraben (blau). Der Graben kann durchwatet werden, dabei ist die Bewegung aber verlangsamt. Als nächstes folgt die undurchdringliche (gelbe) Burgmauer. Im Zentrum des Burghofs liegt der Schatz als goldenes Quadrat. Der Spieler beginnt oben links in der Ecke als orangefarbener Grafikpunkt-Abenteurer. Wenn er den Schatz im Zentrum berührt, ist das Spiel beendet:

```
100 GRAPHICS 19
105 COLOR 3
110 PLOT 11,3
120 DRAWTO 28,3
130 DRAWTO 28,20
140 DRAWTO 11,20
150 DRAWTO 11,3
160 PLOT 12,4
170 DRAWTO 27,4
180 DRAWTO 27,19
190 DRAWTO 12,19
200 DRAWTO 12,4
210 PLOT 13,5
220 DRAWTO 26,5
230 DRAWTO 26,18
240 DRAWTO 13,18
250 DRAWTO 13,5
260 COLOR 0
270 PLOT 19,18
280 DRAWTO 19,20
290 PLOT 20,18
300 DRAWTO 20,20
310 COLOR 2
320 PLOT 18,8
330 DRAWTO 16,8
340 DRAWTO 16,15
350 DRAWTO 23,15
360 DRAWTO 23,8
370 DRAWTO 21,8
380 COLOR 1
390 PLOT 19,11
400 PLOT 20,11
```

```
410 PLOT 19,12
420 PLOT 20,12
500 REM DAS SPIEL BEGINNT
510 X=0:Y=0
520 COLOR 1:PLOT X,Y:M=0:N=0
530 S=STICK(0)
540 COLOR 0:PLOT X,Y
550 IF S=15 THEN 520
560 IF S=7 THEN X=X+1:M=-1
570 IF X>39 THEN X=0
580 IF S=11 THEN X=X-1:M=+1
590 IF X<0 THEN X=39
600 IF S=13 THEN Y=Y+1:N=-1
610 IF Y>23 THEN Y=0
620 IF S=14 THEN Y=Y-1:N=+1
630 IF Y<0 THEN Y=23
640 LOCATE X,Y,F
650 IF F=0 THEN 520
660 IF F=1 THEN 700
670 IF F=2 THEN X=X+M:Y=Y+N:GOTO 520
680 FOR W=0 TO 400:NEXT W
690 GOTO 520
700 GRAPHICS 18
710 POSITION 5,4
720 ? #6;"GESCHAFFT!"
730 IF PEEK(53279)=6 THEN 100
740 GOTO 730
```

100: Grafik 3 ohne Textfenster.

105 bis 250: ziehen einen drei Grafikpunkte breiten quadratischen Burggraben und

260 bis 300: löschen einen Teil davon als Übergang.

310 bis 370: zeichnen die Burgmauer in Gelb.

380 bis 420: Jetzt fehlt nur noch der orangene Schatz im Zentrum des Burghofes.

Für eine solche PLOT-Arbeit ist es unerläßlich, eine Skizze anzufertigen. Am einfachsten ist, Sie kopieren sich das Bildschirmraster von Seite 44 (für die GR.3-Auflösung oder von Seite 78 für die feineren Betriebsarten). So können Sie die gesuchten Spalten- und Zeilenwerte direkt ablesen. Sie ersparen sich damit viel Ärger, denn ob alle Daten stimmen, sehen Sie ja erst, wenn das Bild aufgebaut wird, und das bedeutet u.U., das Programm wieder und wieder zu starten, um immer neue Korrekturen an den Grafikdaten vorzunehmen. Bei längeren Programmen ist es auf jeden Fall ratsam, die

Richtigkeit der Daten blockweise durch einen Probelauf des Programms zu überprüfen. Bei diesem Programm hieße das, nach Zeile 420 eine Zeile 425 GOTO 425 einzugeben, damit das Bild nicht vom Programmende gelöscht wird.

500: Nachdem das Spielfeld gezeichnet ist kommt der Held ins Spiel, der in die Burg eindringen und den sagenhaften Schatz retten soll (ob er will oder nicht).

510: setzt X und Y auf 0, die fürderhin die Position des Grafikpunktes (Spieler) angeben werden. Wie wir gleich in

520: sehen werden. Denn in Farbe 1 (Orange) wird hier ein Grafikpunkt in X,Y gePLOTtet. Ferner werden die Variablen M und N auf 0 gesetzt, die sich bald noch als ausgesprochen nützlich erweisen sollen.

530: fragt Stock(0) ab und verfrachtet seinen Wert in S.

540: Hier wird doch unser Spieler in der Hintergrundfarbe ausgerechnet an der Stelle gePLOTtet, wo er eben noch in seiner goldenen Rüstung stand. Das macht ihn ganz schön nervös, und er beginnt zu flackern, denn

550: WENN S=15 (in Ruhe) DANN nach 520, wo der Grafikpunkt erneut gezeichnet wird. An der alten Stelle, denn die X- und Y-Werte haben sich inzwischen ja nicht verändert.

Wenn Sie das Flackern des Spielers stört, dann werden Sie später noch andere Möglichkeiten kennenlernen, die verhindern, daß der Spieler einen Rattenschwanz von Grafikpunkten hinter sich herzieht, wenn er loszieht. Dies ist die unkomplizierteste Methode: Bevor die Position des Spielers verändert wird, löscht ihn Farbe 0 aus, nach der Veränderung wird er neu gePLOTtet.

560: Jetzt geht wieder die Abfragerei des Stöckchens los. Das ist Ihnen ja inzwischen bestens geläufig. (?) Aber was ist das: M=-1! In dieser Variablen wird die alte Position des Spielers für alle Fälle aufbewahrt. Man kann ja nie wissen. Hat X einen Schritt +1 gemacht, notiert M einen Schritt -1.

570: Wenn der Spieler am rechten Rand des Grafikfensters weiter nach rechts gesteuert wird (X größer 39) dann kommt er links (X gleich 0) wieder ins Bild.

580 und 590: Die Bewegung nach links ist entsprechend gestaltet.

600: Bei der Bewegung nach unten bewahrt N die alte Position.

610 bis 630: entsprechen der Regelung für X sinngemäß.

640: bevor der Spieler die neue Position bePLOTten darf, wird nun erst einmal vorgetastet. LOCATE sieht sich an, welch' Farbe Kind das Feld ist, auf das es den Spieler drängt. In der Variablen F wird der COLOR-Wert der Position X,Y notiert.

650: WENN F gleich 0 ist, DANN ist alles in Ordnung. Das neue Feld ist Hintergrund, darf also betreten werden. Ab gehts nach 520, wo der Schritt vollzogen wird.

660: WENN F gleich 1 ist, DANN hat der Spieler den Orange-Schatz gefunden. Das Spiel ist vorbei. Ab nach 700 dem Ende entgegen.

670: WENN F gleich 2 ist, versucht der Spieler gerade, mit dem Kopf durch die Burgmauer zu rennen. Das können wir nicht zulassen. Zum Glück haben wir in M und N die alte Position des Spielers aufgeschrieben. Wir versetzen die Koordinaten wieder in den alten Stand (X=X+M und Y=Y+N) und führen nach 520 zurück, wo der Spieler auf X,Y gePLOTtet wird und das ist genau die Stelle, von der aus er die Burgmauer angerannt hat.

680: Wenn alle vorherigen Bedingungen nicht zutrafen, dann kann F nur noch den Wert 3 haben. Es gibt keine andere Möglichkeit. Deshalb brauchen wir das nicht extra abzufragen, sondern können gleich loslegen, wenn das Programm überhaupt bis in diese Zeile vorgedrungen ist. Wenn F gleich 3 ist, dann steht der Spieler bis zum Hals im Burggraben. Da kann er natürlich nicht so flink rumflitzen wie auf dem trockenen Land. Also programmieren wir eine Warteschleife, indem wir den Rechner mal eben bis 400 zählen lassen und dann erst nach 520 zurückspringen. Da dort die vom Spieler verlassene Position in der Hintergrundfarbe gePLOTtet wird, schlägt das Durchwaten eine Gasse in die Fluten des Burggrabens. Wenn Ihnen das biblisch vorkommt, dann sollten Sie sofort in die Tasten greifen und entsprechende Umprogrammierungen einleiten.

700: Läutet das Ende ein. Grafik 2 ohne Textfenster wird angeknipst und in

710: an die POSITION 5,4

720: ge?et (und zwar per Adresse Grafikfenster!), daß es vollbracht ist.

730: fragt die Adresse ab, die sich mit den Funktionstasten verbunden fühlt. Steht dort zufällig eine 6, dann ist gerade die START-Taste gedrückt und dann gehts nach 100 auf ein Neues. Ist START nicht gedrückt, dann wird in Zeile

740: nach 730 zurückgesprungen. In diesem Kreislauf bleibt das Programm hängen, bis sich irgendwann jemand erbarmt und die START-Taste doch drückt oder den Computer ausschaltet.

Wenn die Spielfelder unseres Programms der Einteilung des Grafikfensters entsprechen und alle spielwichtigen Sachverhalte durch wechselnde Farben dargestellt werden können, dann kann sich die Spielfigur mit dem LOCATE-Befehl auf sehr unkomplizierte Weise wie mit einem Blindenstöckchen durch das Spielfeld tasten.

Es mag aber geschehen, daß diese schlichten Möglichkeiten unseren hochtrabenden Spielideen nicht genügen. Unsichtbare Fallgruben z.B. sind auf diese Weise nicht programmierbar. Ich möchte Ihnen deshalb noch die doppelt indizierte Variable als Schiedsrichter vorstellen. Diese Variable, nennen wir sie einmal F(X,Y) bekommt ein Fach für jedes Spielfeld. Über die Bildschirmposition 9,12 wacht die Variable F(9,12) usw. Mit der Positon des Spielers (X,Y) haben wir auch immer die entsprechende Variable (F(X,Y)) zur Hand.

Im folgenden Spiel finden Sie sich als rotes Quadrat am linken Bildschirmrand wieder. Ihre Aufgabe: Verlassen Sie das Grafikfenster auf der rechten Seite! Zwischen Ihnen und dem Ziel liegt nur schwarze Finsternis. Wenn Sie kühn voranstürmen, ecken Sie bald an. Das Hindernis ergraut, und Sie müssen wohl (oder übel) Ihren Weg ändern. Weil Sie Ihren Weg als Blutspur hinter sich herziehen, blinkt der Spieler wieder einmal, damit er sich im Meer seiner Spuren nicht verliert:

```
10 DIM F(39,23)
20 GRAPHICS 18
30 POSITION 6,4
40 ? #6;"VORSPANN"
50 FOR A=0 TO 39
60 FOR B=0 TO 23
70 H=0
80 IF RND(0))0.69 THEN H=3
90 F(A,B)=H
100 NEXT B
110 NEXT A
200 GRAPHICS 19:POKE 710,2:POKE 708,50
210 X=0:Y=11
220 M=0:N=0
230 COLOR 1:PLOT X,Y
240 COLOR 0:PLOT X,Y
250 COLOR 1:PLOT X,Y
```

```
260 K=STICK(0)
270 IF K=7 THEN X=X+1:M=-1
280 IF X>39 THEN 400
290 IF K=11 THEN X=X-1:M=1
300 IF X<0 THEN X=0:M=0
310 IF K=13 THEN Y=Y+1:N=-1
320 IF Y>23 THEN Y=23:N=0
330 IF K=14 THEN Y=Y-1:N=1
340 IF Y<0 THEN Y=0:N=0
350 IF F(X,Y)=0 THEN GOTO 220
360 COLOR 3:PLOT X,Y
370 X=X+M:Y=Y+N
380 GOTO 220
400 FOR Y=0 TO 23
410 FOR X=0 TO 39
420 IF F(X,Y)=3 THEN COLOR 3:PLOT X,Y
430 NEXT X
440 NEXT Y
450 FOR P=0 TO 500:NEXT P
460 GRAPHICS 35:POKE 710,2:POKE 708,50
470 POKE 752,1:POKE 82,0
480 ? "GESCHAFFT! - ";
490 GOTO 480
```

10: DIMensioniert für jede Bildschirmposition eine Variable.

20: GRAPHICS 18 ein.

30 und 40: bringen einen Vorspann auf die Mattscheibe, denn das Einlesen von 960 Variablenwerten fordert schon etwas Geduld. In diesem Vorspann könnten Erläuterungen zu dem Spiel abgegeben werden. Während der Benutzer den Text studiert, läuft das Programm unbemerkt weiter.

50 und 60: setzen die Schleifenzähler für zwei verschachtelte FOR-NEXT-Schleifen, die das Einlesen der Werte in die doppelt indizierte Variable F besorgen.

70: H soll 0 sein. Aber wenn in

80: der Zufall es so will, dann wird H auf 3 gesetzt. Rein statistisch bleibt H in 69% aller (Zu-) Fälle gleich 0 und in den restlichen 31% wird sie 3.

90: Der doppelt indizierten Variablen wird der Wert H zugewiesen.

100 und 110: beenden diesen Programmblock. Per Zufall ist in der Variablen F(n,n) ein Spielfeld geschaffen, es hat also bei jedem neuen Spiel eine andere Form. Etwa ein Drittel aller Werte ist 3 und wird als Hindernis behandelt.

200: Das Spiel beginnt. Grafik und Farbe werden bestimmt.

210: Die Anfangswerte X und Y für den Spieler.

220: M und N, zwei alte Bekannte, werden auf 0 gesetzt.

230 bis 250: lassen die aktuelle Position des Spielers flackern, damit er sich nicht in seiner eigenen Spur verirrt. Wenn Sie diese beiden Zeilen löschen und spielen, werden Sie merken, was ich damit meine.

280 bis 340: fragen in gewohnter Weise STICK(0) ab. In

280: wird nach 400 gesprungen, wenn X größer 39 ist, denn damit ist der rechte Rand erreicht, das Spiel beendet.

350: Wenn die Schiedsrichter-Variable 0 enthält, dann darf der Spieler das angestrebte Feld betreten. Das Programm wird nach 220 zurückgeführt und der Schritt vollzogen. Sonst

360: bleibt nur die Möglichkeit, daß F(X,Y) den Wert 3 enthält, also ein Hindernis darstellt. Mit COLOR 3 wird die Barriere sichtbar gemacht und in

370: werden die alten Werte von X und Y durch M und N wieder hergestellt. Wenn dann in

380: nach 220 zurückgesprungen wird, dann befindet sich der Spieler noch an der alten Stelle, nämlich vor der jetzt sichtbaren Wand.

400: wird nur erreicht, wenn die Siegbedingung (X größer 39) erfüllt ist.

410 bis 440: mit einer verschachtelten FOR-NEXT-Schleife wird das in F(X,Y) gespeicherte Spielfeld sichtbar gemacht.

450 bis 490: blenden nach einem Päuschen das Textfenster ein und drucken endlos, daß es geschafft sei.

Ein abschließendes Wort noch zum Spielfeld. Die Hindernisse werden von RND bestimmt. Und wenn es der Zufall will, mag es passieren, daß kein Weg zum Ziel frei bleibt. Die Wahrscheinlichkeit dafür ist allerdings gering. Experimentieren Sie ein wenig mit dem Wert in Zeile 80. Wenn Sie ihn verringern, ist bei 0,6 fast schon kein Durchkommen mehr. Bei einem größeren Wert (0,8) wird die Aufgabe so leicht, daß es keinen Spaß mehr macht. Erfahrungswerte.

4.3 Die Feinde

Der Computer kann einen menschlichen Gegenspieler nicht ersetzen, denn er kann nicht wirklich Entscheidungen treffen. Der Rechner kann nur in mechanischer Weise auf vorab definierte Situationen in vorbestimmter Weise reagieren. Er bringt keine Gefühle ins Spiel, kann also kein Risiko eingehen, bluffen, sich irren, schlecht spielen oder gut in Form sein. Er ist immer gleich und das wird schnell langweilig.

Wie können wir diesem Dummkopf Züge einbläuen, die wenigstens eine bescheidene Illusion von Intelligenz vermitteln? Wir müssen für die Züge des Elektronengehirns eine Strategie programmieren. Aber selbst für ganz einfache Aufgaben entsteht dabei schnell ein Wust von Programmzeilen. Die Überlegenheit der Maschine ergibt sich nur aus der unfehlbaren Präzision und der mit menschlichem Maßstab gemessen beliebig steigerbaren Geschwindigkeit.

In den folgenden Abschnitten werden drei Prinzipien der Steuerung vorgestellt. In der einfachsten Version laufen die (Computer-) Feinde in festen Bahnen. Viele bekannte Computerspiele (Space Invaders, Centipede) basieren auf dieser schlichten Methode.

Differenzierter wird das "Verhalten" der Gegner, wenn durch zufällige Elemente der Anschein von Spielentscheidungen erweckt wird. So können die Feinde z.B. zufällig ihre Richtung oder Geschwindigkeit wechseln.

Bei der feinsten Methode (angewendet z.B. bei Pac Man) werden die Bewegungen der Gegenspieler errechnet. Mit einer Suchroutine hält der Computer Ausschau nach dem Spieler und wenn er ihn entdeckt hat, lenkt er seine Schritte in die entsprechende Richtung.

4.3.1 Auf ausgetretenen Pfaden

Der Spieler steht mit dem Rücken zur Wand. Er kann nur nach rechts oder links laufen. Der Feind kommt in Wellen auf ihn zu, von oben. Jeder hat seine feste Bahn, also die X-Position bleibt immer gleich, der Y-Wert nimmt stetig zu. Der Spieler wird mit dem Joystick bewegt. Und schließlich müssen noch die Punkte gezählt werden.

```
100 GRAPHICS 19
110 H=19
120 DIM X(39),Y(39)
130 FOR I=0 TO 39
140 X(I)=0:Y(I)=0
150 NEXT I
200 IF RND(0)>0.33 THEN 230
210 I=INT(RND(0)*40)
220 IF Y(I)=0 THEN Y(I)=1:X(I)=I
230 FOR I=0 TO 39
240 COLOR 1:PLOT X(I),Y(I)
250 NEXT I
300 REM DER SPIELER
310 COLOR 0:PLOT H,23
320 Q=STICK(0)
330 R=STRIG(0)
340 H=H+((H<39) AND (Q=7))*(R+1)
360 H=H-((H>0) AND (Q=11))*(R+1)
380 COLOR 3:PLOT H,23
400 REM DIE FEINDE
410 FOR I=0 TO 39
420 IF Y(I)>0 THEN Y(I)=Y(I)+1
430 IF Y(I)=23 THEN IF X(I)<>H THEN GOSUB 800
440 IF Y(I)=23 THEN IF X(I)=H THEN GOSUB 700
450 NEXT I
480 IF PEEK(53279)=5 THEN 500
490 GOTO 200
500 GRAPHICS 35:POKE 752,1
510 ? "    Sie haben ";U;" Feinde gefangen"
520 ? :? "    Aber ";V;" sind Ihnen entwischt"
530 GOTO 530
700 U=U+1
```

```
710 GOSUB 900
720 RETURN
800 V=V+1
810 GOSUB 900
820 RETURN
900 COLOR 0
910 PLOT X(I),0
920 DRAWTO X(I),23
930 X(I)=0:Y(I)=0
940 RETURN
```

110: die vertikale Anfangsposition des Spielers

120: X und Y werden DIMensioniert und

130 bis 150: auf 0 gesetzt.

200: Jetzt gehts los. Wenn der Zufall es will, wird nach 230 gesprungen. Sonst gelangt das Programm nach

210: wo I einen Zufallswert zwischen 0 und 39 erhält.

220: Wenn auf der Bahn X(I) noch kein Feind unterwegs ist, dann ist der Wert Y(I) noch 0 und wenn das so ist, dann wird jetzt ein Angreifer losgeschickt. X(I), die Spalte, bekommt den Wert I und Y(I), die Zeile, den Wert 1.

230 bis 250: PLOTten alle Angreifer auf allen Bahnen. Weil hier ohne Rücksicht auf Verluste alle X(I),Y(I) gePLOTtet werden, aber immer einige den Wert 0,0 haben, erscheint in der Ecke oben links ein Grafikpunkt, der da nichts zu suchen hat. Nur um diesen Störenfried wegzubekommen, müßte das Programm einige Zeilen länger sein. Wenn Sie das der Mühe wert finden, bitteschön!

300: Jetzt kommt der Spieler zum Zuge.

310: Zuerst wird er weggePLOTtet.

320 bis 360: Jostick und Feuerknopf bewegen ihn ein oder zwei Positionen nach rechts oder links.

380: Dann wird er wieder hingePLOTtet.

400: Die Feinde stürmen los.

420: Ist ein Angreifer auf der Bahn, dann ist sein Y-Wert größer 0 und dann wird dieser Y-Wert jetzt um 1 vergrößert, d.h. er macht einen Schritt nach unten.

430: Und nun müssen wir natürlich aufpassen, ob er schon unten angekommen ist (Y(I)=23). WENN er unten angekommen ist DANN - WENN der Spieler nicht in der Bahn steht, DANN geht es ins Unterprogramm nach Zeile 800.

440: Aber WENN der Spieler zur Stelle ist, dann gehts im Unterprogramm von Zeile 700 weiter.

480: fragt die SELECT-Taste (PEEK(53279)=5) ab. Ist sie gedrückt, wird das Spiel abgebrochen und das Ergebnis ausgegeben. Ist sie nicht gedrückt, dann geht es in Zeile

490: zurück zum nächsten Durchlauf.

500 bis 530: zelebrieren das Ende des Spiels und verkünden das Ergebnis.

700: War der Spieler zur Stelle, wenn der Feind unten ankam, bekommt er mit U=U+1 einen Punkt.

710: Dann gehts ins Unter-Unterprogramm.

800: War der Spieler nicht zur Stelle, bekommt der Computer mit V=V+1 einen Punkt.

810: Dann gehts ebenfalls ins Unter-Unterprogramm.

900 bis 940: In diesem Unter-Unterprogramm wird mit der Farbe des Hintergrundes die gesamte Bahn des erledigten Feindes gelöscht. Danach springt das Unter-Unterprogramm in das Unterprogramm zurück und das Unterprogramm springt ins Hauptprogramm zurück.

4.3.2 Schicksalsschläge

Feinde, die in festgefahrenen Bahnen daherkommen, sind nicht besonders gefährlich, obwohl sich auch aus dieser simplen Programmstruktur viel herausholen läßt (wenn Sie nur wollen).

Jetzt aber führt der Computer den Zufall ins Feld. Die Wege der Feinde werden dadurch unergründlich. Das ist besonders herausfordernd, wenn man ihn einfangen soll. Da das Programm sehr kurz ist, flattert der Feind wie eine Motte hin und her. Der Spieler zieht als rotes Quadrat los und kann graue Netze hinter sich lassen, in denen sich der Falter fängt. Auch diese Programmstruktur ist ausbaufähig:

```
100 GRAPHICS 19
110 POKE 708,50:POKE 709,2:POKE 710,20
120 XC=1:YC=11
130 XU=39:YU=11
140 COLOR 1:PLOT XU,YU
200 COLOR 0:PLOT XC,YC
210 IF RND(0)<0.66 THEN 230
220 M=0:N=0:C=INT(RND(0)*4)
230 IF C=0 THEN IF XC<38 THEN XC=XC+1:M=-1
240 IF C=1 THEN IF XC>1 THEN XC=XC-1:M=1
250 IF C=2 THEN IF YC>1 THEN YC=YC-1:N=1
260 IF C=3 THEN IF YC<22 THEN YC=YC+1:N=-1
270 LOCATE XC,YC,H
280 IF H<>0 THEN XC=XC+M:YC=YC+N:GOTO 220
290 COLOR 3:PLOT XC,YC
300 COLOR 0:PLOT XU,YU
310 S=STICK(0)
320 T=STRIG(0)
330 IF T=0 THEN COLOR 2:PLOT XU,YU
340 IF S=15 THEN GOTO 200
350 XU=XU+((S=7) AND (XU<39))
360 XU=XU-((S=11) AND (XU>0))
370 YU=YU+((S=13) AND (YU<23))
380 YU=YU-((S=14) AND (YU>0))
390 COLOR 1:PLOT XU,YU
400 GOTO 200
```

100 und 110: Grafik und Farbwerte.

120: Die Ausgangswerte für den Computer-Feind

130: Die Ausgangswerte für den User-Spieler.

140: PLOTtet den Spieler in Farbe 1.

200: Hier beginnt der Computer, seines Weges zu ziehen. Da wird er erst einmal weggePLOTtet. Dann wird eine neue Position für ihn errechnet. Das geschieht in

210: Damit der dumme Rechner nicht immer nur im Kreise läuft, wird seine Marschrichtung nur etwa jedes dritte Mal geändert. Experimentieren Sie mit anderen Werten.

220: Wenn es der Zufall so will, dann wird hier auch per Zufall eine neue Richtung erkoren. M und N stehen wieder zur Stelle.

230 bis 260: ändern die Koordinatenwerte je nach gefundenem Richtungsumschwung. Beim ersten Durchlauf ist der Wert von C mit größerer Wahrscheinlichkeit 0. C=0 führt den Feind nach rechts. Das erscheint sinnvoll, weil er am linken Rand das Spiel beginnt.

270: Doch bevor dem Elektronengehirn ein Schritt erlaubt wird, muß er mit der LOCATE-Brille nachsehen, ob das Feld, auf das es ihn zieht, den Farbwert 0 (Hintergrund) hat.

280: Findet er einen Wert ungleich 0, dann hat der Spieler hier bereits seine Netze ausgeworfen und der Weg ist versperrt. Die Koordinaten des Feindes werden mit M und N auf den alten Wert zurückgesetzt. Und nun muß sich der Rechner für einen anderen Weg "entscheiden". Damit dieser "Entscheidungsprozeß" keinen Tempoverlust für den dummen Computer bedeutet, erlauben wir ihm solange einen Versuch, bis er endlich einen Schritt machen konnte. (Man will ja nicht so sein.) Also ab nach 220.

290: Hat er nun endlich einen Weg gefunden, wird gePLOTtet.

300: Jetzt ist der Spieler am Zug. Auch er wird erst einmal weggePLOTtet.

310 und 320: fragen Stöckchen und Knöpfchen ab.

330: Ist der Trigger gepreßt, wird mit Farbe 2 ein Hindernis an der aktuellen Stelle des Spielers hinterlegt.

340: Wenn der Knüppel in Ruhe steht, kommt sofort der Computer wieder zum Zug.

350 bis 380: fragen die verschiedenen Stellungen des Joysticks ab und verändern die Koordinatenwerte.

390: druckt den Spieler in seiner Farbe auf den Bildschirm

400: und dann gehts in die nächste Runde.

Natürlich hat der User leichtes Spiel. Er braucht ja nur den feurigen Knopf gedrückt zu halten und immer engere Schlingen um den schlingernden Flatterfalter zu ziehen, bis der hilflos auf der Stelle zappelt.

Falls Sie es aber auf den vergangenen 195 Seiten noch nicht gemerkt haben sollten, dieses Buch liefert Ihnen (fast) keine Fix-und-Fertig-Aptipp-Stumpfsinn-Programme. Ich möchte Sie mit verschiedenen Programmiertechniken und Spielprinzipien vertraut machen und entsprechende Programmstrukturen vorstellen. Ich hatte mir das dann so gedacht, daß Sie aus diesem Angebot komplexere Spiele in ansprechender Ausstattung mit Bedienungskomfort und allem Drum und Dran selbst machen können (wenn Sie wollen).

Bei diesem Spiel wäre es denkbar, dem Spieler nur einen begrenzten Vorrat an Hindernissen mit auf den Weg zu geben oder das Setzen eines Hindernisses mit einem Zeitverlust zu verbinden. Das Programm könnte auch einfach nur mitzählen, wie viele Hindernisse gebraucht wurden, um den Falter zu erhaschen oder die Zeit messen, die es gebraucht hat (s. Kapitel 4.4). Aber wie ich Sie inzwischen kenne, haben Sie selbst noch viel bessere Ideen, was man aus dieser Spielidee alles herausholen kann. Ich bin jedenfalls drauf gefaßt, daß Sie mein schönes Prograämmlein völlig umkrempeln.

4.3.3 Gnadenlose Präzision

Bislang sind die Feinde blind in vorgezogenen Bahnen gelaufen, haben wirr herumgetastet. Jetzt aber erhält der Feind ein unfehlbar wachsames Auge, und er wird mit der ihm eigenen Exaktheit und Unfehlbarkeit seinen Weg wählen. So wie der Spieler auch in jedem Moment seine Figur und die Figur des elektronischen Widersachers im Auge hat, so lassen wir nun auch den Computer mit seinem elektronischen Auge die Lage überblicken.

In den Schaltkreisen der Maschine geistern die aktuellen Koordinaten des Spielers herum. Wir müssen ihr nur erlauben, diese Daten auszuwerten. Im folgenden Spiel bekommt ATARI zwei Figuren und die Aufgabe, den Spieler damit zu fangen. Der Spieler läuft weg, so gut er kann. aber wohin er sich auch wenden mag, die Feinde sind ihm unbeirrbar auf den Fersen:

```
100 GRAPHICS 3:POKE 752,1
110 POKE 708,188:POKE 709,50
120 SX=19:SY=9
130 AX=0:AY=0
140 BX=39:BY=19
200 COLOR 1:PLOT SX,SY
210 COLOR 2:PLOT AX,AY:PLOT BX,BY
220 IF ABS(AX-SX)<ABS(AY-SY) THEN 250
230 IF AX-SX<0 THEN P=1:GOTO 270
240 IF AX-SX>0 THEN P=-1:GOTO 270
250 IF AY-SY<0 THEN Q=1
260 IF AY-SY>0 THEN Q=-1
270 LOCATE AX+P,AY+Q,L
280 IF L=1 THEN 500
290 COLOR 0:PLOT AX,AY
300 AX=AX+P:AY=AY+Q:P=0:Q=0
310 IF ABS(BX-SX)<ABS(BY-SY) THEN 340
320 IF BX-SX<0 THEN P=1:GOTO 360
330 IF BX-SX>0 THEN P=-1:GOTO 360
340 IF BY-SY<0 THEN Q=1
350 IF BY-SY>0 THEN Q=-1
360 LOCATE BX+P,BY+Q,L
370 IF L=1 THEN 500
```

```
380 COLOR 0:PLOT BX,BY
390 BX=BX+P:BY=BY+Q:P=0:Q=0
400 M=STICK(0)
410 IF M=15 THEN 200
420 COLOR 0:PLOT SX,SY
430 SX=SX+((SX<39) AND (M=7))
440 SX=SX-((SX>0) AND (M=11))
450 SY=SY+((M=13) AND (SY<19))
460 SY=SY-((M=14) AND (SY>0))
490 GOTO 200
500 ? :? "            G E F A N G E N"
510 GOTO 500
```

100: Grafik einschalten, Cursor unterdrücken

110: Farben definieren.

120: Die Anfangsposition des Spielers und

130 und 140: die der beiden Feinde.

200: Den Spieler PLOTten und

210: die Feinde auch.

220: Jetzt wird der Weg für Feind A errechnet. Als erstes stellt das Programm fest, ob A vom Spieler in X- oder in Y-Richtung weiter entfernt ist. Dazu wird die Differenz (= Entfernung) der X-Koordinaten von Feind und Spieler und der Y-Koordinaten gegenübergestellt. Da diese Differenz positiv oder negativ sein kann, je nach Lage von Spieler und Feind, aber die Entfernung gesucht ist, interessiert das Vorzeichen ob negativ oder positiv nicht. Nur der ABSolute Betrag ist für uns von Interesse. Der Befehl ABS eleminiert das Signum (Vorzeichen), während der Befehl SGN nur das SiGNum findet.

230: Wenn sich der Feind A links vom Spieler befindet (die X-Koordinate von A ist kleiner als die X-Koordinate des Spielers) dann wird in der Variablen P eine +1 notiert.

240: Befindet sich A rechts vom Spieler, dann wird mit P=-1 ein Schritt nach links vorbereitet.

250: War die Y-Entfernung größer, dann wird hier auf die gleiche Art ein Schritt nach unten

260: oder nach oben ermittelt.

270: Jetzt lokalisiert der Rechner die anvisierte Stelle und wenn er dort den (COLOR-) Wert 1 entdeckt,

280: dann hat er den Spieler entdeckt - ab nach 500.

290: Jetzt wird Feind A weggePLOTtet, dann

300: werden seine Koordinaten um den errechneten Wert (P oder Q) verändert und die hilfreichen Variablen P und Q für den nächsten Einsatz wieder gelöscht. Neu gePLOTtet wird Feind A am Anfang des nächsten Programmdurchlaufs.

310 bis 390: erledigen die exakt gleiche Arbeit für Feind B.

400: Jetzt darf der Spieler mit dem Knüppel zuschlagen.

410: Stock in Ruhe, schon ist der Rechner wieder am Ball.

420: Der Spieler wird weggePLOTtet,

430 bis 460: ermitteln seine neuen Koordinaten und

490: führt zum nächsten Durchlauf zurück, wo Feinde und Spieler sogleich ins Bild gesetzt werden.

500: War ein Sprung hierher möglich, dann ist das Spiel aus.

4.4 Zeichen der Zeit

Wie das Leben selbst, so muß auch das Spiel ein Ende haben. Ein Spiel kann beendet werden, indem die Siegbedingung erfüllt wird oder eine vorab vereinbarte Zeit abgelaufen ist. Bei Computer-Spielen wird in Ermangelung eines menschlichen Gegners gern die verrinnende Zeit zum bösen Feind aufgebaut. In diesem Kapitel sollen Sie erfahren, wie die Spieldauer definiert und der Ablauf der Zeit dargestellt werden kann.

Nur so zum Spaß eine Stoppuhr:

```
100 GRAPHICS 19:DEG
110 POKE 708,2:POKE 709,52
120 FOR W=540 TO 210 STEP -30
130 X=SIN(W)*7+19:Y=COS(W)*7+11
140 COLOR 1:PLOT 19,11:DRAWTO X,Y
200 FOR WW=540 TO 210 STEP -15
210 S=S+1:IF S=24 THEN S=0
220 XX=SIN(WW)*11+19:YY=COS(WW)*11+11
230 COLOR 2:PLOT 19,11:DRAWTO XX,YY
240 IF PEEK(53279)=5 THEN 500
250 FOR P=0 TO 138:NEXT P
260 COLOR 0:PLOT 19,11:DRAWTO XX,YY
270 COLOR 1:PLOT 19,11:DRAWTO X,Y
280 NEXT WW
300 COLOR 0:PLOT 19,11:DRAWTO X,Y
310 Z=Z+1
320 NEXT W
330 GOTO 120
500 GRAPHICS 35:POKE 710,0:POKE 752,1
510 POKE 708,50:POKE 709,52
520 ZEIT=Z*10+S/2.4
530 ? "              ";ZEIT
540 IF PEEK(53279)=6 THEN 100
550 GOTO 540
```

Der große Zeiger macht einen Umlauf in 10 Sekunden. Jeder der 24 Takte zählt also 0,42 Sek. Geeicht wird er durch die Warteschleife in Zeile 250. Durch Druck auf die SELECT-Taste wird die Uhr angehalten und die Zeit ausgedruckt.

4.4.1 Zeitgerecht

Die einfachste Methode der Zeiterfassung ist eine Variable, die bei jedem Programmdurchlauf fortgezählt wird (Z=Z+1). Nach einer vorprogrammierten Anzahl von Durchläufen (Z größer n), wird das Spiel unterbrochen. Dabei muß natürlich verhindert werden, daß der Spieler den Programmablauf durch INPUT o.ä. unterbrechen kann. Die Abarbeitungszeit des Programmes ist die Maßeinheit für die Spieldauer:

```
100 REM ZPLUSZ.BIR
110 GRAPHICS 3
120 FOR P=0 TO 1000
130 NEXT P
140 Z=Z+1
150 IF Z<36 THEN 110
160 ? "ENDE";
170 GOTO 170
```

120 und 130: enthalten eine leere Schleife. An dieser Stelle steht das Spielprogramm.

140: zählt die Durchläufe.

150: Hier wird die Zeit-Variable abgefragt. Solange sie den Wert 36 noch nicht erreicht hat, wird mit dem Rücksprung nach 110 eine neue Runde eingeläutet. Sonst geht es in

160: weiter bzw. zuende.

Der ATARI verfügt aber auch über eine eingebaute elektronische Uhr. Für diese Funktion sind die Register 18, 19 und 20 vorgesehen. Beim Einschalten des Rechners oder nach einem Warmstart (RESET) beginnen alle drei Register mit dem Wert 0. Das Register 20 zählt von 0 bis 255. Wenn es dann wieder auf 0 umspringt, wird ein Zählimpuls an das Register 19 abgegeben. So zählt 19 seinerseits von 0 bis 255, um beim Umschlagen einen Impuls an Register 18 zu senden. Den Zeitwert der internen Uhr erfahren wir also mit der Anfrage PEEK(18)*256*256 + PEEK(19)*256 + PEEK(20). Als höchster Wert kann mithin 16.777.215 dargestellt werden.

Die Uhr tickt im 50-Hertz-Takt, also Register 20 zählt in einer einzigen kleinen Sekunde von 0 bis 49. Die interne

Uhr kann danach 335.544 Sekunden oder 5592 Minuten oder ca. 93,2 Stunden (knapp vier Tage) erfassen. Falls Sie bei einem Spiel die Zeit nehmen wollen, das länger läuft, müssen Sie entsprechende Programm-Vorkehrungen treffen. Allerdings wäre vorher zu überlegen, ob man dem Rechner eine solche Dauerbelastung zumuten sollte:

```
100 REM INTUHR.BIR
110 ? CHR$(125)
120 POKE 18,0:POKE 19,0:POKE 20,0
130 A=PEEK(20)
140 B=PEEK(19)
150 C=PEEK(18)
160 POKE 752,1
170 T=A+256*B+256*256*C
180 IF T>9 THEN P=1
190 IF T>99 THEN P=2
200 IF T>999 THEN P=3
210 IF T>9999 THEN P=4
220 IF T>99999 THEN P=5
230 POSITION 20-P,11
240 ? T
250 GOTO 130
```

110: Bildschirm löschen.

120: Bevor Sie das Programm eingetippt haben, ist die interne Uhr schon bei 5.000 und mehr, deshalb setzen wir sie erst einmal auf 0. Sie fängt dann sofort wieder an zu zählen. Sie werden also im Register 20 nie eine 1 finden. Bis sie dort nachgesehen haben, ist schon ein höherer Wert darin. Bei diesem Programm ist die Uhr sogar schon bei 8, bevor der erste Ausdruck erfolgt.

130 bis 150: Die drei Register der internen Uhr werden ge-PEEKt.

160: Der Cursor wird versteckt.

170: errechnet den numerischen Wert aus den drei Registern und dann könnte er auf den Bildschirm gedruckt werden. Mit einem POS.-Befehl wird aber die Stelle auf der Mattscheibe angesteuert, bei der der Ausdruck beginnt. Die auszugebende Zahl ist aber erst einstellig, später aber mehrstellig. Weil es schöner aussieht, wenn die letzte Stelle der gedruckten Zahl immer an der gleichen Bildschirmposition steht, müssen wir feststellen, wie lang die aktuelle Zahl ist. Das geschieht in den Zeilen

180 bis 220: in der Variablen P, die nur auf sechs Stellen

vorbereitet ist. Wenn Sie über eine Million zählen lassen wollen, müssen Sie entsprechend weitere Zeilen einfügen.

280: Jetzt können wir die exakte POS. angeben, vor der beginnend gePRINTet werden soll. Der Wert von P ist um so höher, je mehr Stellen die zu druckende Zahl hat. Da wir diesen Wert bei der POS.itionierung abziehen, beginnt die Ausgabe einer längeren Zahl entsprechen weiter vorne in der Zeile.

240: T wird gePRINTet.

250: Dann eilen wir zurück, um zu sehen, was die Stunde nun geschlagen hat.

4.4.2 Zeitdruck

Selbst ein schlichtes Computer-Spielchen macht Eindruck, wenn die verrinnende Zeit auf der Mattscheibe sichtbar wird. Nun wollen wir uns als Profi-Heim-Programmierer nicht damit abgeben, einfach die numerischen Wert aus der internen Uhr oder unserer Zählvariablen (womöglich noch im Textfenster) auszugeben. Ein bißchen grafischer Hokus-Pokus sollte schon der Mühe wert sein:

```
100 REM ZEITMESS.BIR
110 GRAPHICS 2
120 POKE 710,0:POKE 752,1
130 FOR P=0 TO 100
140 NEXT P
150 Z=Z+1
160 IF Z=10 THEN 200
170 GOTO 130
200 Z=0
210 M=M+1
220 ? CHR$(18);
230 IF M=36 THEN 300
240 GOTO 120
300 POSITION 8,3
310 ? #6;"ENDE"
320 GOTO 320
```

110: Grafik mit Textfenster.

120: Das Textfenster bekommt die Farbe des Hintergrundes, weil das eleganter aussieht. Außerdem wird der Cursor weggePOKEt.

130 und 140: Hier gehört das Spielprogramm hin.

150: zählt die Durchläufe und

160: veranlaßt bei jedem zehnten einen Abstecher nach 200.

170: Ansonsten gehts wieder von vorne los.

200: Wenn Z in 160 den Wert 10 hatte, dann wird sie hier als erstes auf 0 gestutzt.

210: Dann wird mitgezählt, wie oft sich das Programm in diese Gefilde verlaufen hat (M=M+1) und

220: das ATASCII-Zeichen 18 gePRINTet. Das ist ein waagerechter Strich. Das ; kümmert sich darum, daß der (unsichtbare) Cursor gleich hinter dem gePRINTeten Zeichen verharrt und bei nächsten Mal dort weiterdruckt, so daß nach und nach ein schöner, langsam wachsender Balken am unteren Bildschirmrand wächst.

230: Wenn der Balken 36 Zeichen lang ist (rechts bleiben zwei Schreibpositionen frei und links auch) dann gehts nach Zeile 300 zum Abpfiff.

240: Und sonst kommt der nächste Durchlauf dran.

300: Das Spiel ist aus. In die schon eingeschaltete GR.2

wird in Zeile 3 ab Spalte 8

310: über Datenkanal Nummer 6 das "ENDE" verkündet.

320: Das wars dann.

Statt mit ATASCII-Zeichen läßt sich die Zeit natürlich auch mit Grafikpunkten darstellen. Vom Spielplatz muß dann entsprechend Raum abgeknappst werden. Ein schrumpfender Balken macht das Verrinnen der Zeit noch deutlicher:

```
100 REM ZEITBALK.BIR
110 GRAPHICS 19
130 COLOR 1:PLOT 39,0:DRAWTO 39,23
140 FOR P=0 TO 300
150 NEXT P
160 Z=Z+1
170 COLOR 0:PLOT 39,Z-1
180 IF Z=24 THEN 200
190 GOTO 140
200 GRAPHICS 35
210 POKE 710,0:POKE 752,1
220 ? "                DAS WAR'S!"
230 GOTO 230
```

130: Der Zeitbalken wird am rechten Rand des Grafikfensters gePLOTtet und dann

170: nach den Angaben von Z Stück für Stück weggePLOTtet.

180: Wenn der gesamte Balken verschwunden ist (Z=24),

200 bis 230: wird das Ende angezeigt.

4.5 Wertsachen

Wer von einem hirnlosen Rechner um die Genugtuung betrogen wird, ein menschliches Gegenüber in die Schranken gewiesen zu haben (wenn es auch nur ein Spiel war), der braucht einen anderen Maßstab, seinen Triumph auszukosten. Mit der Maschine wird zumeist um Punkte gespielt, wenn es auch verschiedene Regeln gibt, nach denen diese Punkte gesammelt werden, am Ende zählt, was man zusammenbekommen hat.

Es kann geregelt werden, daß die Punkte im Verlauf des Spiels zu sammeln sind. Bei dem Programmbei-Spiel für Feinde in festen Bahnen wurde gezählt, wie viele Feinde gefangen wurden und wie viele durchkamen. Das Verhältnis aus beiden Zahlen war das Spielergebnis, das man auf die gleiche Weise grafisch darstellen könnte, wie es für die Zeit erklärt worden ist.

Eine Variante wäre es, wenn der Spieler im Verlauf des Spiels bestimmte Punkte auf dem Spielfeld betreten müßte und dafür mit Punkten belohnt würde. Mit dem LOCATE-Befehl können wir kontrollieren, ob der entsprechende Punkt erreicht wurde und in einer Wertungsvariablen würde der Spielstand mitgezählt. Das Prinzip besteht also darin, bestimmte Situationen während des Spiels herbeizuführen (Feinde fangen, Spielfelder betreten, Schätze finden), was fortlaufend mitgezählt und am Ende als Ergebnis ausgeworfen wird.

Beim folgenden Spiel sollen Sie mit Ihrem roten Spieler die blauen Grafikpunkte - und damit Punkte einsammeln. Es erscheint immer nur ein blauer Punkt auf dem Spielfeld. Sie gleiten mit dem Joystick hin, heimsen den Punkt ein und schon erscheint irgendwo ein neuer Punkt.

Dummerweise wächst das ganze Spielfeld mehr und mehr mit grauen Hindernissen zu. Und am unteren Rand des Grafikfensters schwillt kontinuierlich ein Zeitbalken, während die im Textfenster angezeigten Punkte nur spärlich mehr werden.

Sammeln Sie so viele Punkte wie möglich, bevor die Zeit abgelaufen ist:

```
100 GRAPHICS 3:POKE 708,52:POKE 709,144
110 POKE 710,2:POKE 752,1
115 X=19:Y=9:COLOR 1:PLOT X,Y
120 COLOR 3:PLOT 0,19:DRAWTO 39,19
130 U=INT(RND(0)*40):V=INT(RND(0)*19)
140 COLOR 2:PLOT U,V
150 IF RND(0)<0.85 THEN 180
160 A=INT(RND(0)*40):B=INT(RND(0)*19)
170 COLOR 3:PLOT A,B
180 S=STICK(0):M=0:N=0
190 IF S=15 THEN 400
200 IF S=7 THEN X=X+1:M=-1
210 IF X>39 THEN X=39:M=0
220 IF S=11 THEN X=X-1:M=1
230 IF X<0 THEN X=0:M=0
240 IF S=13 THEN Y=Y+1:N=-1
250 IF Y>18 THEN Y=18:N=0
260 IF S=14 THEN Y=Y-1:N=1
270 IF Y<0 THEN Y=0:N=0
280 LOCATE X,Y,F
290 IF F=2 THEN GOSUB 600
300 IF F=3 THEN X=X+M:Y=Y+N:M=0:N=0
310 COLOR 1:PLOT X,Y
320 COLOR 0:PLOT X+M,Y+N
400 Z=Z+1:IF Z=30 THEN Z=0:ZZ=ZZ+1
410 COLOR 2:PLOT ZZ,19
420 ? ,,W;CHR$(28)
430 IF ZZ=39 THEN ? W;" ist Ihr Endergebnis";:GOTO 500
440 SOUND 0,0,0,0:GOTO 150
500 GOTO 500
600 W=W+1:SOUND 0,14,14,8
610 U=INT(RND(0)*40):V=INT(RND(0)*19)
620 LOCATE U,V,H:IF H<>0 THEN 610
630 COLOR 2:PLOT U,V
640 RETURN
```

100 und 110: Grafik, Farben, Cursorunterdrückung.

115: Die Startkoordinaten für den Spieler, der auch gleich gePLOTtet wird.

120: Mit Farbe 3 wird direkt über dem Textfenster ein Balken gezogen. Da das Textfenster die gleiche Farbe hat, bilden beide eine farbliche Einheit.

130: Für den Sammelpunkt werden zufällige Koordinaten U und V bestimmt

140: und in seiner blauen Farbe gleich ausgedruckt.

150: Wenn die hier zufällig generierte Zahl kleiner als 0,85 ist, dann wird ein Sprung nach 180 angeordnet. Das ist in 85% aller Fälle zu erwarten.

160: Nur bei etwa jedem siebten Durchlauf gelangt das Programm deshalb hierher und erzeugt zufällige Koordinaten A und B, die

170: in Farbe 3 als Hindernis gePLOTtet werden.

180 bis 270: Die übliche Abfragerei des Steuerknüppels.

280: LOCATE überwacht die Züge des Spielers.

290: Wenn der Wert 2 gefunden wird (Sammelpunkt), dann wird in der Subroutine ab 600 eingesammelt.

300: Wenn der Spieler ein Hindernis vor den Füßen hat, dann werden seine alten Koordinatenwerte wieder hergestellt.

310: Sodann wird der Spieler auf seine neu berechnete Position gezeichnet

320: und mit Hintergrundfarbe die alte Position weggetüncht. Bei dieser Vorgehensweise, erst die neue Position des Spielers zu drucken und dann die alte fortzunehmen, flimmert der Spieler nicht so nervös, verschwindet dafür aber für etwas länger, wenn er sich nicht bewegt.

400: Hier wird die Zeit gezählt. Z zählt mit den Programmdurchläufen von 0 bis 30 und setzt jedesmal, wenn 30 erreicht ist ZZ um 1 höher.

410: Da sich der Variablenwert ZZ nur bei jedem 30sten Durchlauf ändert, bringt der PLOT mit Farbe 3 auch nur dann eine sichtliche Veränderung. Ein Grafikpunkt wird auf den Zeitbalken gePLOTtet.

420: PRINTet den laufenden Punktestand in das Textfenster.

430: Wenn der Zeitbalken den rechten Bildschirmrand erreicht hat (ZZ=39), wird das Endergebnis (W) ausgedruckt und das Programm in Zeile 500 aufgefangen.

440: schaltet den Tongenerator 0 aus (im nächsten Kapitel erfahren Sie mehr darüber), der weiter unten im Programm erst eingeschaltet wird und springt nach 150 zurück.

600: Das Unterprogramm. Ein Punkt wurde eingesammelt. Hier wird er von der Variablen W gezählt und ein Ton wird angestimmt (der dann beim nächsten Durchlauf in Zeile 440 wieder ausgeknipst wird. Durch diese Anordnung haben wir mehr Zeit, auch etwas davon zu hören.)

610: Koordinaten für einen neuen Sammelpunkt werden erwürfelt, dann

620: wird überprüft, ob die Stelle auf dem Bildschirm frei ist und dann erst

630: mit Farbe 2 gePLOTtet.

640: RETURN nicht vergessen.

Natürlich ist es möglich, daß ein Sammelpunkt im Verlauf des Spiels an eine Stelle gesetzt wird, die vom Spieler nicht mehr erreicht werden kann, obwohl noch Zeit übrig ist. Pech!

Das Programm hat aber auch noch eine kleine Macke, die sich aber mit einer Zeile beheben läßt, und diese Arbeit habe ich für Sie aufgespart. So wie das Programm jetzt läuft, kann es nämlich passieren, wenn es der dumme Zufall so will, daß ein Hindernis genau auf den Sammelpunkt gePLOTtet wird. Dann hat der Spieler plötzlich nichts mehr zu gewinnen. Sie müssen also eine Zeile schreiben, die verhindert, daß ein Hindernis an die Stelle gedruckt wird, an der sich gerade der Sammelpunkt befindet. Diese Anweisung gehört in Zeile 165.

Anregungen für die Erweiterung dieses Spiels: Lassen Sie den Sammelpunkt nach einer wohl dosierten Zeit den Ort wechseln oder setzen sie ihn (mit dem Zufallsmotor) in Bewegung. Lassen Sie den Sammelpunkt von Zeit zu Zeit unsichtbar werden. Oder geben Sie zwei oder mehr Punkte auf einmal aus.

Die laufende Wertung eines Spiels kann auch einfach darin bestehen, die Schritte zu zählen, die zur Lösung der Aufgabe benötigt wurde. Bei Ratespielen bietet sich diese Wertung besonders an, da des Rätsels Lösung üblicherweise in möglichst wenigen Schritten gefunden werden soll.

Es ist aber auch möglich, das Punkteergebnis erst am Ende des Spiels auszuzählen. Beim folgenden Programm huschen Spieler und Gegenspieler (für zwei Personen) über den Bildschirm und färben ihn in ihrer jeweiligen Farbe ein. Natürlich überpinselt der eine, was der andere gerade für sich beansprucht hat. Am Ende wird ausgezählt, wer die größere Fläche bekleckert hat.

Weil ich sicher bin, daß Sie keine Schwierigkeiten haben werden, die Joystick-Abfrage für den zweiten Spieler in das folgende Programm einzufügen, liefere ich Ihnen die Version, in der Sie gegen den Rechner anmalen können:

```
100 GRAPHICS 19:POKE 708,52:POKE 709,144
110 POKE 710,4:POKE 752,1
115 COLOR 3:PLOT 0,23:DRAWTO 39,23
120 XS=0:YS=0:COLOR 1:PLOT XS,YS
130 XC=39:YC=22:COLOR 2:PLOT XC,YC
140 IF RND(0)<0.8 THEN 160
150 C=INT(RND(0)*4)
160 XC=XC-((C=0) AND (XC>0))
170 XC=XC+((C=1) AND (XC<39))
180 YC=YC-((C=2) AND (YC>0))
190 YC=YC+((C=3) AND (YC<22))
200 COLOR 2:PLOT XC,YC
210 D=D+1:IF D<2 THEN 140
220 D=0
230 S=STICK(0)
240 XS=XS+((XS<39) AND (S=7))
250 XS=XS-((XS>0) AND (S=11))
260 YS=YS+((YS<22) AND (S=13))
270 YS=YS-((YS>0) AND (S=14))
280 COLOR 1:PLOT XS,YS
300 Z=Z+0.05
310 COLOR 0:PLOT 39-Z,23
320 IF Z=39 THEN 400
330 GOTO 140
400 GRAPHICS 35:POKE 710,4:POKE 752,1
405 POKE 82,0:POKE 708,52:POKE 709,144
410 ? "         Es folgt die Wertung"
420 ? :? "       Bitte haben Sie etwas Geduld";
430 FOR X=0 TO 39
440 FOR Y=0 TO 22
450 LOCATE X,Y,E
460 IF E=1 THEN SE=SE+1
470 IF E=2 THEN CE=CE+1
480 NEXT Y
490 NEXT X
500 ? :? :? "        Sie haben ";SE;" Felder erobert"
510 ? :? "   Der Computer hat ";CE;" Felder besetzt";
520 GOTO 520
```

115: In der untersten Zeile des Grafikfensters wird ein Balken gePLOTtet, der künftig die Zeit darstellen wird.

120: Der Spieler.

130: Der Computer.

140: Es folgt die "zufällige Steuerung" (ein merkwürdiges Wort, nicht wahr? Zufall oder Steuerung?) In 80% aller Durchläufe behält der Computer seine Bewegungsrichtung (C)

bei , aber bei etwa jedem fünften Mal

150: bestimmt er per RND eine neue Richtung.

160 bis 190: besorgen die Veränderung der Koordinaten des Computers ja nach Richtung C.

200: Dann wird der neue Punkt XC,YC in der Farbe des Computers (COLOR 2) gedruckt. Weil der Rechner auf diese Weise recht dumm über den Bildschirm stolpert und wir ihm keine weitere Strategie einprogrammieren wollen, nach der er möglichst viel Land gewönne,

210: ersetzen wir Klasse durch Masse. Der Computer darf bei jedem Durchlauf zwei Felder weit ziehen.

220: D wird für den nächsten Durchlauf auf 0 gesetzt.

230: Jetzt ist der Spieler an der Reihe. Sein STICK(0) wird abgefragt.

240 bis 270: verändern entsprechend die Koordinaten des Spielers und

280: druckt ihn aus.

300: Eine vereinfachte Form der Zeitnahme. Die Variable Z wird bei jedem Durchlauf nur um Dezimalteile erhöht.

310: Da der PLOT-Befehl Deziamlstellen wegrundet, wird erst dann ein neuer Punkt im Zeitbalken sichtbar, wenn Z um eine ganze Stelle größer geworden ist. In diesem Programm braucht es dafür zwanzig Durchläufe.

320: Wenn Z den Wert 39 erreicht hat (nach 800 Durchläufen) dann gehts nach 400.

330: Nächster Durchlauf bitte.

400: Das Textfenster wird eingeblendet.

405: Nach einem GR.-Befehl müssen die POKEs erneuert werden.

420: Die Wertung braucht wirklich etwas Geduld, denn

430 bis 490: gehen mit einer verschachtelten Schleife jeden einzelnen Grafikpunkt durch und lesen seinen COLOR-Wert. Je nach Farbe wird dann für den Spieler (SE=SE+1) oder den Computer (CE=CE+1) ein Pünktchen notiert.

500 und 510: Drucken das ermittelte Endergebnis ins Textfenster.

520: Schluß.

4.6 Der gute Ton

Bei Science-Fiction-Filmen kracht und rumst es selbst im schalltoten Weltraum, denn die Kinomacher wissen, daß der richtige SOUND ein wichtiges Bestandteil der Illusion ist. Das gilt für Computer-Spiele nicht minder.

Die Grafik, die uns der Hausfreund in BASIC zu bieten hat, ist zu grob, um naturalistische Visionen damit zu erzeugen oder zu schwerfällig, dabei auch in Schwung zu bleiben. Die Player-Missile-Grafik bietet in dieser Beziehung mehr, aber da verirren wir uns auch schon fast in den aufgedampften Schaltkreisen des Chip.

Ein paar gekonnte Töne machen jedoch aus jedem roten Rechteck einen schnittigen Rennwagen, die Berührung mit einem Hindernis knistert elektrisch und eingesammelte Punkte klingeln in der Kasse.

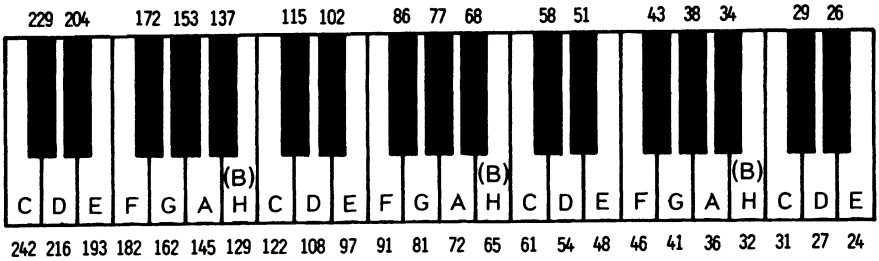
ATARI ist stimmgewaltig. Mit vier getrennten Tongeneratoren bringt er den Lautsprecher des Monitors/Fernsehers in Stimmung. Mit nur einem BASIC-Wort können hunderte von Tönen und Geräuschen erzeugt und zu endlosen Klangbildern zusammenprogrammiert werden.

Mit SOUND (Abkürzung SO.) werden die Generatoren und Töne definiert. Hinter SOUND folgen vier Kennzahlen, die durch Kommata getrennt werden. Der Befehl sieht so aus:

SOUND (Generator),(Tonhöhe),(Verzerrung),(Lautstärke)

Einmal angestimmt dröhnt ein Tongenerator so lange, bis der Strom abgeschaltet wird, der Befehl END erfolgt oder ein anderer Ton für den gleichen Generator bestimmt wird. Dabei kommt die Lautstärke 0 dem Ausschalten des Generators gleich.

Es können also vier verschiedene Töne/Geräusche gleichzeitig erzeugt werden. Aber begnügen wir uns fürs erste mit einem. Die Generatoren werden durch die Kennzahlen 0 bis 3 angesprochen. Die Tonhöhe reicht von 0 (hoch) bis 255 (tief). Diese 256 Werte ergeben reine Töne, die allerdings der klassischen Tonleiter nur bei bestimmten Werten und nur näherungsweise entsprechen. Die Abbildung stellt eine Ver-



bindung zwischen der Tonleiter und diesen Tonwerten her.

Der Verzerrungsgrad wird durch eine gerade Zahl von 0 bis 14 bestimmt. 0 ergibt die übelste Verzerrung, 14 läßt den Ton rein erklingen (10 übrigens auch). Mit den Verzerrungen lassen sich viele interessante Spielgeräusche erzeugen. Sie dürfen sich übrigens nicht wundern, daß bestimmte Kombinationen von Tonwerten und Verzerrungsgraden gar keinen Laut von sich geben. Das liegt in der Natur der Verschieberegister und sonstiger Delikatessen, die Teile des SOUND-Generators sind*.

Die vierte Kennzahl für die Lautstärke kann Werte von 0 bis 15 bekommen. 0 entspricht dem Ausschalten, 1 ist leise und 15 laut.

Jede der vier Kennzahlen kann durch Variablen oder zu berechnende Werte ersetzt werden.

Jeder einzelne Ton besteht also aus drei Werten und bedarf außerdem einer Kennzahl für den Generator, dem er bestimmt ist. Die Länge der Töne wird dadurch bestimmt, wann sie wieder abgeschaltet werden. Der folgende Ton löscht den vorherigen. Zwischen beiden Tönen kann eine Warteschleife die Dauer des gerade eingeschalteten Tones bestimmen. Die Länge der Warteschleife wäre ein vierter Wert für den Ton. Ein ganzes Lied zu programmieren ist, ähnlich wie bei der Grafik, reine Fleißarbeit. Wer Programmieren will, muß zählen können wie ein Maikäfer.

Rein rechnerisch können 256 (Töne) * 8 (Verzerrungsgrade) * 15 (Lautstärken) = 30.720 Töne, oder sagen wir besser Geräusche, erzeugt werden. Im Zusammenklang mehrerer Tongeneratoren und von FOR-NEXT-Schleifen moduliert, lassen sich damit unglaubliche Effekte erzielen.

* Wie Sie Ihre StrATARIvadi millionenfach erklingen lassen, POKEt Ihnen Alfred Görgens in seinem "ATARI Sound- und Musik-Buch", das in der gleichen Reihe erschienen ist.

Wenn Sie sich einen Ton anhören wollen, dann geben Sie ihn als Direkteingabe ein, also ohne Zeilennummer nur den Befehl SO. r,t,k,l und RETURN-Taste. Der Ton steht und ATARI grüßt mit READY. Jetzt können Sie mit dem Cursor zurückfahren und eine Kennzahl ändern. RETURN. Und schon hören sie den geänderten Ton. Natürlich können Sie auf diese Weise auch alle vier Generatoren gleichzeitig tönen lassen.

Für Spiele bieten sich grundsätzlich zwei Anwendungsmöglichkeiten an. Einmal können wir Erkennungs- oder Siegesmelodien erklingen lassen. Da der Computer dann aber den Musikblock des Programmes abarbeitet, kann solange nichts (oder nicht viel) anderes ablaufen. Bewegte Grafik und melodische Tonfolgen so kurz hintereinander laufen zu lassen, daß der Eindruck der Gleichzeitigkeit entsteht, ist ein mühsames Geschäft.

Die zweite Anwendung liegt in kurzen Spielgeräuschen, die entweder so schnell abgearbeitet werden, daß das Programm davon scheinbar nicht unterbrochen wird, oder die nur aus einem Geräusch bestehen, das an einer Stelle des Programms in den Tongenerator gesetzt und in irgendeiner späteren Zeile wieder gelöscht wird.

4.6.1 Klang-Daten

Jeder einzelne Ton wird durch vier Zahlenwerte definiert. Wir müssen den Generator (G) bestimmen, die Tonhöhe (T), den Verzerrungsgrad (R) und die Lautstärke (V). Natürlich können Sie beliebige Variablennamen dafür erfinden.

Ein Ton braucht aber auch eine bestimmte Dauer, die er kingt. Den Generatoren können wir damit nicht kommen. Wir müssen also einen Weg finden, den gesetzten Ton nach einer bestimmbaren Zeit wieder abzuschalten. Bei einer Melodie kann das dadurch geschehen, daß der nächste Ton in den Generator gesetzt wird. Zwischen beiden Tönen muß dann nur eine Pause definierter Länge eingegeben werden. Die Länge der Pause ist ein fünfter Wert, der von uns programmiert werden muß.

Das Programm, mit dem die Töne aufgerufen werden, ist ganz kurz. Der Aufwand steckt in den vielen, vielen Zahlenwerten, die gefunden, geprüft und eingegeben werden müssen:

```
10 TRAP 70
20 READ G,T,R,V,L
30 SOUND G,T,R,V
40 FOR D=0 TO 450/L:NEXT D
50 GOTO 10
70 END
100 DATA 0,129,12,2,900
110 DATA 1,72,14,4,2
120 DATA 1,81,14,3,8
130 DATA 1,65,14,5,2
140 DATA 1,72,14,4,8
150 DATA 0,137,12,1,900
160 DATA 1,77,14,4,2
170 DATA 1,86,14,3,8
180 DATA 1,68,14,5,2
190 DATA 1,77,14,4,8
200 DATA 0,122,12,3,900
210 DATA 1,108,14,5,2
220 DATA 1,122,14,4,8
230 DATA 1,102,14,6,2
```

```
240 DATA 1,108,14,5,4
250 DATA 0,137,12,2,900
260 DATA 1,115,14,3,2
270 DATA 1,129,14,2,8
280 DATA 1,102,14,4,2
290 DATA 1,115,14,3,1
```

10: TRAP ist eine Falle, eine Falle für Fehler. Mit TRAP wird bestimmt, in welcher Zeile weitergearbeitet werden soll, wenn ein Fehler auftritt. Der Rechner bricht dann die Bearbeitung des Programmes nicht ab, sondern macht bei der hinter TRAP genannten Zeilennummer weiter. Wo im folgenden Programm ein Fehler zu erwarten ist, erzähle ich Ihnen später.

20: Statt für jeden einzelnen Ton einen SOUND-Befehl zu schreiben, verstauen wir die entsprechenden Zahlenwerte in DATA- (Daten-) Zeilen und lesen (READ) sie hier in die Variablen ein. Der READ-Befehl bewirkt, daß für die hinter ihm stehenden Variablen eine nach der anderen aus den DATA-Zeilen Werte eingelesen werden. Mit READ können auch String-Variablen gefüllt werden, dann müssen in der DATA-Zeile Zeichenketten stehen. Wo die DATA-Zeilen im Programm stehen, ist ziemlich beliebig. Wenn der erste READ-Befehl ergeht, sucht sich der Rechner die DATA-Zeile mit der niederigsten Zeilennummer, liest die geforderte Menge Daten und merkt sich, an welcher Stelle er die DATA-Zeile verlassen hat. Wenn der nächste READ-Befehl ergeht, macht er dann genau an der Stelle weiter.

30: Mit den in die Variablen G, T, R und V geREADeten DATA wird hier ein SOUND gesetzt.

40: Die Variable L soll bestimmen, wie lange der Ton erklingt. Eine Pause von 0 bis 450 entspricht ungefähr einer ganzen Note. Als Quotient kann L mit den Werten 1, 2, 4, 8 usw. ganze, halbe, viertel, achtel usw. Noten bewirken.

50: löst den nächsten Durchlauf aus. READ holt die folgenden DATA, SOUND läßt sie erklingen und die FOR-NEXT-Schleife beschneidet die Tondauer.

70: END sorgt dafür, daß am ENDe auch die Tongeneratoren ausgeknipst werden.

100 bis 290: enthalten die Daten für zwanzig Töne. Nur der besseren Übersicht wegen stehen die Daten für einen Ton in einer DATA-Zeile. Dem READ-Befehl würde es nichts ausmachen, wenn so viele Daten hintereinander stünden, wie eine BASIC-Zeile verkraften kann. Die Daten dieses Pro-

gramms könnten also in zwei Zeilen untergebracht werden. Das ist schon deshalb sinnvoll, um die vielen DATA-Wörter einzusparen.

Die hier bezifferte Melodie hat zwei Stimmen. Auf Generator 0 laufen dunkle Töne. Damit sie zusammen mit den Tönen aus Generator 1 zu hören sind, wird die FOR-NEXT-Pause umgangen, indem für die Tonlänge ein Wert bestimmt wird, der kleiner als 1 ist. Die Dauer des Tones in Generator 0 wird durch die Gesamtdauer der vier Töne in Generator 1 mitbestimmt, denn erst wenn diese vier Töne abgespielt sind, bekommt Generator 0 neue DATAs.

Wenn Sie gute Nerven haben, dann schreiben Sie in Zeile

```
70 RESTORE :GOTO 10
```

Der Befehl RESTOR bewirkt, daß der Zeiger, der auf das als nächstes zu lesende DATA zeigt, zurückgesetzt wird. Das nächste READ findet dann den ersten Wert in der ersten DATA-Zeile. Auf diese Weise kann man Daten immer wieder abfragen. Unser kleines Programm wird druch RESTORE zum Laierkasten.

Und nun bin ich Ihnen noch trap, trap TRAP schuldig. Durch den Rücksprung in 50 nach 10 läuft das kleine Tonprogramm an sich endlos. Nach dem zwanzigsten Durchlauf sind aber alle Daten gelesen. Wenn jetzt noch ein READ-Befehl ergeht, dann wehrt sich der Rechner mit der ERROR-Feige: ERROR- 6 (zu wenige Daten). Mit der Fehlerfalle bauen wir dem vorausschauend vor und sagen, wenn ein Fehler auftritt, dann spring nach 70 und beENDe das Spiel.

Natürlich kann der Rechner nicht wissen, welchen Fehler wir meinen. TRAP wirkt beim ersten Fehler, der auftritt. Wir müssen schon dafür sorgen, daß nur gewollte Fehler im Programm enthalten sind.

Das kleine Programm hätte auch anders beendet werden können, z.B. mit Nullen als letzte DATAs und einer Abfrage, WENN (z.B.) G=0 DANN ENDE.

4.6.2 Klang-Bilder

Was uns täglich in den Ohren dröhnt, hat selten viel mit Musik zu tun. Aber Geräusche sind auch nur Töne. Mit den Verzerrungsmöglichkeiten und ein paar pfiffigen Manipulationen lassen sich Klangbilder zaubern, von denen die Wiener Philharmoniker nur alpträumen können. Das können Sie nun ausprobieren!

Ich programmiere mir zum Abschied ein Geräuschklavier. Das Keyboard des ATARI läßt rund 256 verschiedene Tasten (Kombinationen) zu. Ein paar, wie z.B. die BREAK-Taste, fallen aus. Trotzdem sind das mehr als genug, denn mit jedem Tastendruck könnte ein anderes Geräusch ausgelöst werden. Wie das funktionieren soll? So:

```
10 OPEN #1,4,0,"K:"
20 GET #1,T
30 CLOSE #1
40 IF (T<49) OR (T>52)=1 THEN 10
50 GOSUB T*100
60 GOTO 10
70 END
4900 FOR T=255 TO 0 STEP -1
4910 SOUND 0,T,8,15
4920 FOR L=0 TO 15:NEXT L
4930 NEXT T
4940 SOUND 0,0,0,0
4950 RETURN
5000 FOR L=6 TO 0 STEP -0.2
5010 FOR V=1 TO 15
5020 SOUND 1,36,14,V
5030 NEXT V
5040 FOR V=2*L TO 1 STEP -1
5050 SOUND 1,40,10,V
5060 NEXT V
5070 NEXT L
5080 SOUND 1,0,0,0
5090 RETURN
5100 FOR M=1 TO 4
5110 FOR V=2 TO 3*M
5120 SOUND 3,2*M,4,V
```

```
5130 FOR P=0 TO V*M*3:NEXT P
5140 NEXT V
5150 NEXT M
5160 SOUND 3,0,0,0
5170 RETURN
5200 SOUND 0,97,14,8
5210 FOR P=0 TO 100:NEXT P
5220 SOUND 1,72,14,7
5230 FOR P=0 TO 200:NEXT P
5240 SOUND 2,145,14,5
5250 FOR P=0 TO 150:NEXT P
5260 SOUND 3,46,14,10
5270 FOR P=0 TO 600:NEXT P
5280 FOR G=0 TO 3:SOUND G,0,0,0:NEXT G
5290 RETURN
```

10: Ich eröffne einen Datenkanal zur Tastatur,

20: lese den ATASCII-Wert der gedrückten Taste in T ein

30: und schließe den Datenkanal wieder.

40: Ich habe mich fürs erste mit den Tasten 1 (ATASCII 49) bis 4 (ATASCII 52) begnügt. Wenn eine andere Taste gedrückt wird, als Programmierer muß man ja auf jeden Unfug vorbereitet sein, dann wird nach 10 zurückgeführt und auf die nächste Eingabe gewartet.

50: Ist eine zugelassene Taste gedrückt worden, wird ihr Wert jetzt mit 100 multipliziert und als Sprungadresse in eine Subroutine verwendet

60: schließt den Programmkreis.

70: grenzt das Hauptprogramm gegen die Subroutinen ab, ist hier aber eigentlich nicht notwendig, weil durch den Rücksprung in 60 diese Zeile gar nicht erreicht werden kann.

4900: War Taste 49 gedrückt, wird hier ein Geräusch angestimmt, das bei

4940: beendet wird.

4950: RETURN-Rücksprung ins Hauptprogramm.

5000 bis 5090: bei Taste 50.

5100 bis 5170: bei Taste 51.

5200 bis 5290: bei Taste 52.

Erfinden Sie selbst weitere Geräusche und vollenden Sie das Geräuschklavier. Experimentieren Sie mit den Programmstrukturen und staunen Sie über die Töne.

5 Appendix

Programme

5.1 Zahlenraten

5.2 Malpinsel

5.3 Farben-Schlüssel

5.4 Bit-Muster

Tabellen

GRAPHICS 1 und 2, numerische Werte für COLOR

Zeichensatz - ATASCII-Code - interner Code

BASIC-Wörter mit Kurzerklärung und Seitenreferenz

5.1 Zahlenraten

Dieses Programm hatte ich Ihnen in Abschnitt 4.1.1 als Hausaufgabe gestellt. Falls Sie die ausnahmsweise mal gemacht haben sollten, bevor Sie nun hier nachschlagen, werden Sie feststellen, wie verschieden Programme aussehen können, die letztlich doch das gleiche leisten. Bei meiner Version können Sie eine Zahl zwischen 1 und 10.000 raten:

```
10 ? CHR$(125)
20 R=INT(RND(0)*9999)+1
30 ? "Bitte geben Sie Ihre Zahl ein"
40 TRAP 200
50 ?
60 INPUT V
70 IF V<R THEN ? :? "zu klein"
80 IF V>R THEN ? :? "zu gross"
90 IF V<>R THEN Z=Z+1:GOTO 30
100 ? :? "Richtig geraten"
110 ? :? "Sie haben ";Z;" Versuche gebraucht"
120 ? "Vielen Dank!"
130 END
200 ? :? "Bitte"
210 ? :? "geben Sie Ihre Zahl"
220 ? :? "nur in Ziffern ein"
230 GOTO 50
```

10: Bildschirm löschen

40: Die TRAP-Falle fängt das Programm auf, falls der User (natürlich nur, um uns zu foppen) Strings statt Ziffern eintippt.

70 und 80: vergleichen die getippte Zahl V mit der Computerzahl R.

90: zählt in Z die Anzahl der Rateversuche und verzweigt nach 30, wenn die Lösung noch nicht gefunden wurde, denn sonst wird in

100 bis 120: das Ergebnis ausgegeben.

200 bis 230: Hierhin verzweigt TRAP, eine Ermahnung an den Benutzer.

5.2 Malpinsel

Der Joystick als Pinsel. Wählen Sie mit dem Feuerknopf aus acht Farbtönen aus und bemalen Sie die Flimmerleinwand:

```
100 GRAPHICS 10
110 C=1:X=39:Y=47
120 POKE 705,14
130 POKE 706,250
140 POKE 707,40
150 POKE 708,50
160 POKE 709,82
170 POKE 710,96
180 POKE 711,160
190 POKE 712,2
200 S=STICK(0)
210 T=STRIG(0)
220 IF T=0 THEN C=C+1
230 IF C>8 THEN C=1
235 IF (T=0) AND (S=15)=1 THEN GOSUB 400
250 IF S=15 THEN 200
260 IF S=7 THEN X=X+1
270 IF X>79 THEN X=79
280 IF S=11 THEN X=X-1
290 IF X<0 THEN X=0
300 IF S=13 THEN Y=Y+1
310 IF Y>47 THEN Y=47
320 IF S=14 THEN Y=Y-1
330 IF Y<0 THEN Y=0
340 COLOR C
350 PLOT X,Y*4
360 DRAWTO X,Y*4+3
370 GOTO 200
400 FOR P=0 TO 50
410 NEXT P
420 RETURN
```

100: erlaubt acht Farbtöne für Grafikpunkte und einen für den Hintergrund.

110: C bestimmt später die Farbe. X und Y sind die Start-

koordinaten des Pinsels.

120 bis 190: definieren die acht Farbtöne. Ich habe mich bemüht, aus den 128 aufrufbaren Farbwerten die Töne herauszufischen, die den Grundfarben und Mischfarben erster Ordnung (Farbenlehre nach Itten) möglichst nahekommen. Hinzu kommen noch Weiß und Mittelgrau. Der Hintergrund ist (bleibt) Schwarz. Die Farbwirkung ist von der Einstellung Ihres Bildschirms abhängig.

230: Wenn Sie C auf 0 statt auf 1 zurücksetzen, können Sie auch die Hintergrundfarbe in den "Pinsel" nehmen und damit Bildschirmteile löschen.

235: Wenn der Feuerknopf gedrückt wird, um die Farbe zu wechseln, während der Pinsel in Ruhestellung ist, wird in ein Unterprogramm nach 400 gesprungen. Die Farben wechseln sonst so schnell, daß man gar nicht richtig wählen kann. Wenn der Pinsel in Bewegung ist, soll ein schneller Wechsel möglich sein. Das gibt einen schönen Regenbogeneffekt!

350: Die Bildauflösung in dieser Betriebsart bewirkt längliche Grafikpunkte. Um das auszugleichen, wird jeder Grafikpunkt in Y-Richtung über vier Bildschirmzeilen gezogen. Es wird also nur in jede vierte Zeile (Y*4) gePLOTtet und der Grafikpunkt über vier Zeilen gezogen (Y*4+3). So erhalten wir quadratische Ausgaben, die vier Bildpunkte breit und vier Fernsehzeilen hoch sind.

Die Zeilen 260 bis 330 lassen sich, wem sage ich das, auch platz-, speicher- und arbeitsschonender schreiben. (Boole, oh Boole!):

```
260 REM PLATZ GESPART
270 REM PLATZ GESPART
280 X=X+((X<79) AND (S=7))
290 X=X-((X>0) AND (S=11))
300 Y=Y+((Y<47) AND (S=13))
310 Y=Y-((Y>0) AND (S=14))
320 REM PLATZ GESPART
330 REM PLATZ GESPART
```

5.3 Farben-Schlüssel

Dies ist ein nützliches Programm (Denglisch: Utility). Über das Keyboard können Sie vier Farbtöne beliebig einstellen und am Bildschirm die Wirkung gleichzeitig beobachten. Wenn Sie eine passende Farbkombination eingestellt haben, liefert Ihnen der Rechner auf Tastendruck P wie POKE die Zahlenwerte, die Sie in die Farbregister schreiben müssen.

Das Programm bezieht sich auf die Grafikgänge 3 bis 7. Wo nur zwei Farben zur Verfügung stehen, müssen Sie die Register 709 und 710 unberücksichtigt lassen. Stellen Sie nur einen Farbton für den Hintergrund und alle drei Farbtöne im Grafikfenster gleich ein, um die richtige Wirkung zu bekommen.

Wenn Sie Farben ausgewählt und die POKE-Werte notiert haben, können Sie mit START wieder in die Grafik gehen.

```
100 GRAPHICS 18:POKE 752,1:POKE 712,0:POSITION 6,4
110 ? #6;"COlOrKEy":FOR Z=0 TO 500:NEXT Z
120 GRAPHICS 0:POKE 752,1:POKE 712,16:POKE 710,18
125 POKE 709,8:POSITION 1,0
130 ? "Mit diesem Programm lassen sich Farben"
140 POSITION 6,2:? "durch Tastendruck einstellen"
150 POSITION 6,4:? "und gleichzeitig die Wirkung"
160 POSITION 5,6:? "auf dem Bildschirm beobachten."
170 POSITION 1,10
175 ? "Tasten  1  2  3  4  regeln den Farbton"
180 POSITION 0,12
185 ? "Tasten   Q  W  E  R  die Helligkeit dazu"
190 POSITION 5,15:? "Taste P ruft die POKE-Werte ab"
200 POSITION 12,19:? "Weiter mit START"
210 POSITION 4,22:? "(c) 1984   Clausthalt-Zellerfeld"
220 POSITION 11,23:? "by Karl-Heinz Koch";
230 IF PEEK(53279)=6 THEN 250
240 GOTO 230
250 GRAPHICS 3:POKE 752,1:POKE 712,4:POKE 708,6
260 POKE 709,2:POKE 710,8:S0=0:S1=0:S2=0:S3=0
270 ? "    Farbton        1  2  3  4"
280 ? :? "    Helligkeit      Q  W  E  R";
290 COLOR 1:PLOT 19,6:DRAWTO 19,0:DRAWTO 0,0
```

```
300 POSITION 0,6:POKE 765,1:XIO 18,#6,0,0,"S:"
310 COLOR 1:PLOT 19,18:DRAWTO 19,12:DRAWTO 0,12
320 POSITION 0,18:POKE 765,1:XIO 18,#6,0,0,"S:"
330 COLOR 2:PLOT 39,6:DRAWTO 39,0:DRAWTO 20,0
340 POSITION 20,6:POKE 765,2:XIO 18,#6,0,0,"S:"
350 COLOR 2:PLOT 39,18:DRAWTO 39,12:DRAWTO 20,12
360 POSITION 20,18:POKE 765,2:XIO 18,#6,0,0,"S:"
370 COLOR 3:PLOT 39,11:DRAWTO 39,7:DRAWTO 0,7
375 POSITION 0,11:POKE 765,3:XIO 18,#6,0,0,"S:"
380 OPEN #1,4,0,"K:":GET #1,T
400 IF T=49 THEN C0=C0+1:IF C0>15 THEN C0=0
410 IF T=50 THEN C1=C1+1:IF C1>15 THEN C1=0
420 IF T=51 THEN C2=C2+1:IF C2>15 THEN C2=0
430 IF T=52 THEN C3=C3+1:IF C3>15 THEN C3=0
440 IF T=81 THEN H0=H0+2:IF H0>14 THEN H0=0
450 IF T=87 THEN H1=H1+2:IF H1>14 THEN H1=0
460 IF T=69 THEN H2=H2+2:IF H2>14 THEN H2=0
470 IF T=82 THEN H3=H3+2:IF H3>14 THEN H3=0
480 IF T=80 THEN CLOSE #1:GOTO 540
490 S0=C0*16+H0:POKE 712,S0
500 S1=C1*16+H1:POKE 708,S1
510 S2=C2*16+H2:POKE 709,S2
520 S3=C3*16+H3:POKE 710,S3
530 CLOSE #1:GOTO 380
540 GRAPHICS 0:POKE 752,1:POKE 712,16:POKE 709,8
550 POKE 710,18:POSITION 10,3
555 ? ">  C O L O R  K E Y  <"
560 POSITION 2,5
565 ? "Die POKE-Werte zu den Farbregistern:"
570 POSITION 7,9:? "Hintergrundfarbe   Color 0"
580 POSITION 15,10:? "POKE 712,";S0
590 POSITION 7,12:? "Grafikpunkt        Color 1"
600 POSITION 15,13:? "POKE 708,";S1
610 POSITION 7,15:? "Grafikpunkt        Color 2"
620 POSITION 2,16
625 ? "und Schrifthelligkeit im Textfenster"
630 POSITION 15,17:? "POKE 709,";S2
640 POSITION 7,19:? "Grafikpunkt        Color 3"
650 POSITION 7,20:? "und Farbton im Textfenster"
660 POSITION 15,21:? "POKE 710,";S3
670 IF PEEK(53279)=6 THEN GOTO 250
680 GOTO 670
```

100 bis 240: Vorspann mit Bedienungsanleitung.

250 und 260: Schlichte Grautöne als Anfangsfarben, vier Variablen für die Farbwerte werden auf 0 gesetzt. Wenn Sie diesen Befehl herausnehmen, bleiben die zuletztgewählten

Farbwerte erhalten. Wenn Sie am Ende des Programms, nach Ausgabe der POKE-Werte, die START-Taste drücken, dann erscheint zuerst wieder das graue Grafikbild. Wenn Sie aber irgendeine Taste drücken, erscheinen sofort Ihre alten Farbtöne auf der Mattscheibe.

270 und 280: zeigen im Textfenster an, welche Tasten zu drücken sind.

290 bis 375: malen das Testbild.

380: fragt die Tastatur ab.

400 bis 480: verändert je nach Tastendruck verschiedene Variablen für Farbtöne und Helligkeiten.

490 bis 520: errechnen aus diesen Variablen die POKE-Werte.

530: führt zur erneuten Abfrage der Tastatur zurück.

540: Taste P wurde gedrückt. GRAPHICS 0 löscht das Testbild, alle POKEs müssen erneuert werden.

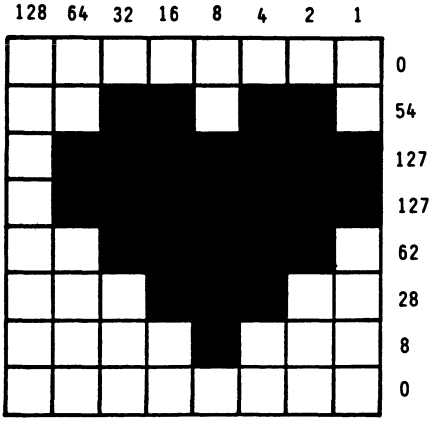
550 bis 660: bauen die Schautafel mit den POKE-Werten auf.

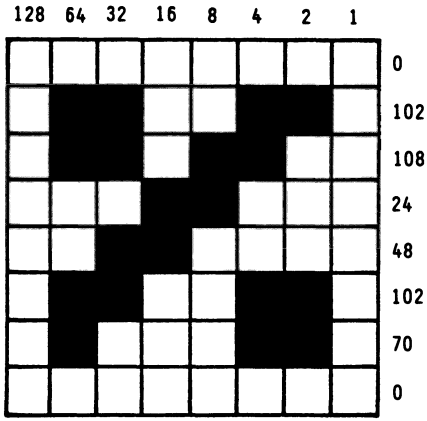
670 und 680: warten, daß START gedrückt wird.

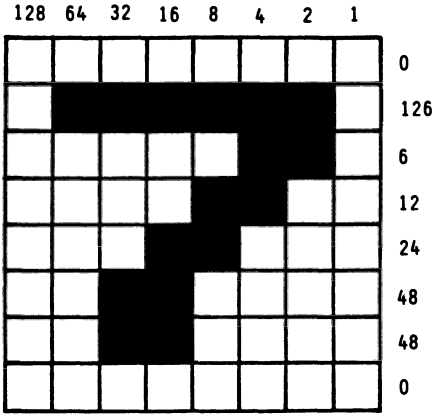
5.4 Bit-Muster

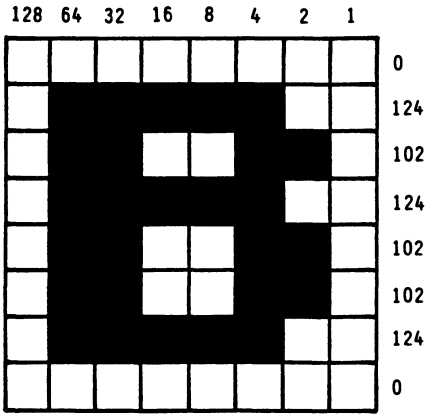
Mit diesem Programm können Sie sich in der Vergrößerung ansehen, wie die Zeichen aus dem ATARI-Zeichensatz aus einzelnen Punkten zusammengesetzt sind. Sie brauchen nur eine beliebige Taste zu drücken und schon erscheint das zugehörige Zeichen auf Grafik-3-Punkte vergrößert auf dem Bildschirm. Nur die übermächtige BREAK-Taste entzieht sich dem Anblick. Aber DELETE, ESC, TAB, CURSOR etc, können Sie sich ansehen:

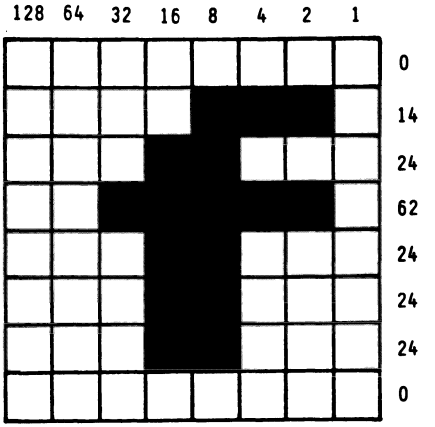
```
10 GRAPHICS 19:DIM F(7,7):POKE 765,2:POKE 712,0
20 POKE 710,0:POKE 709,50:POKE 708,248
30 COLOR 2:PLOT 15,23:DRAWTO 15,0:DRAWTO 0,0
40 POSITION 0,23:XIO 18,#6,0,0,"S:"
50 COLOR 2:PLOT 39,23:DRAWTO 39,0:DRAWTO 24,0
60 POSITION 24,23:XIO 18,#6,0,0,"S:"
70 COLOR 2:PLOT 23,7:DRAWTO 23,0:DRAWTO 16,0
80 POSITION 16,7:XIO 18,#6,0,0,"S:"
90 COLOR 2:PLOT 23,23:DRAWTO 23,16:DRAWTO 16,16
100 POSITION 16,23:XIO 18,#6,0,0,"S:"
110 POKE 765,3:COLOR 3:PLOT 23,15:DRAWTO 23,8
120 DRAWTO 16,8:POSITION 16,15:XIO 18,#6,0,0,"S:"
200 FOR I=0 TO 7:FOR J=0 TO 7:F(I,J)=3:NEXT J:NEXT I
210 OPEN #1,4,0,"K:"
220 GET #1,Z
230 CLOSE #1
240 IF Z<128 THEN POKE 710,0:POKE 708,248
250 IF Z>127 THEN Z=Z-128:POKE 710,248:POKE 708,0
260 IF Z<32 THEN Z=Z+64:GOTO 310
270 IF Z<96 THEN Z=Z-32:GOTO 310
280 IF Z<128 THEN 310
310 FOR I=0 TO 7
320 A=PEEK(57344+I+Z*8)
330 IF A>127 THEN A=A-128:F(0,I)=1
340 IF A>63 THEN A=A-64:F(1,I)=1
350 IF A>31 THEN A=A-32:F(2,I)=1
360 IF A>15 THEN A=A-16:F(3,I)=1
370 IF A>7 THEN A=A-8:F(4,I)=1
380 IF A>3 THEN A=A-4:F(5,I)=1
390 IF A>1 THEN A=A-2:F(6,I)=1
```

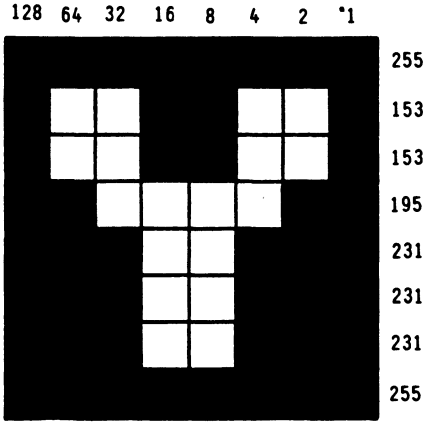

128 64 32 16 8 4 2 1
0
54
127
127
62
28
8
0


128 64 32 16 8 4 2 1
0
102
108
24
48
102
70
0


128 64 32 16 8 4 2 1
0
126
6
12
24
48
48
0


128 64 32 16 8 4 2 1
0
124
102
124
102
102
124
0


128 64 32 16 8 4 2 1
0
14
24
62
24
24
24
0


128 64 32 16 8 4 2 1
255
153
153
195
231
231
231
255

```
400 IF A>0 THEN F(7,I)=1
410 NEXT I
420 FOR J=0 TO 7
430 FOR I=0 TO 7
440 COLOR F(I,J):PLOT I+16,J+8
450 NEXT I
460 NEXT J
500 GOTO 200
```

10: In die Variable F sollen die acht mal acht Bildpunkte gefüllt werden, aus denen sich jedes einzelne Zeichen zusammensetzt.

30 bis 120: bauen eine Grafik auf den Bildschirm.

210 bis 230: fragen die Tastatur ab.

240 bis 280: Die ATASCII-Werte ordnen alle verfügbaren Zeichen in eine Reihe von 0 bis 127 (128 bis 255 sind die gleichen Zeichen nur in negativer Darstellung). Der durch unseren Tastendruck verursachte (ATASCII-) Wert muß umgerechnet werden, weil die Zeichen in einer anderen Reihenfolge (Offspring) im Speicher aufgehoben werden. Das ist der interne Code. Fragen Sie bitte nicht mich, warum das so ist.

Die Daten des Zeichensatzes liegen im ROM-Bereich des Speichers (Read Only Memory, ein Speicherbereich, aus dem man nur lesen (PEEK), in den hinein man aber nicht POKEn kann). Jedes Zeichen besteht aus acht Byte, jedes Byte enthält ein Bit-Muster, das in acht Bildpunkte umgesetzt wird. Der Zeichenspeicher beginnt bei der Adresse 57344. Dort liegt das erste Byte des ersten Zeichens.

310: Die acht Byte eines Zeichens werden einzeln gelesen.

320: A übernimmt den Dezimalwert aus der entsprechenden Adresse, die sich aus Startadresse (57344), dem Byte innerhalb des Zeichens (I) und dem Offspring des Zeichens (Z*8) zusammensetzt.

330 bis 400: brechen den Dezimalwert des Byte in die Werte der acht Bits auf und ordnen die Bit-Werte in die doppelt indizierte Variable F.

410: Wenn das mit acht Bytes gemacht ist, dann befinden sich in der Variable F die acht mal acht Bit, die in Bildpunkte umgesetzt das Zeichen auf die Mattscheibe befördern.

420 bis 460: PLOTten die acht mal acht Bits in ihrer jeweiligen Farbe (0 oder 1) auf den Bildschirm.

Auf ein Wort noch zu Zeile 250. Der Zeichensatz besteht nur aus 128 (0 bis 127) Zeichen, im Speicher liegen also nur 128 mal 8 Dezimalwerte in entsprechend vielen Adressen. Die zweite Hälfte des Zeichensatzes besteht aus den gleichen Zeichen in inverser (negativer) Darstellung. Da müssen nicht noch einmal Daten gespeichert werden, die indirekt schon verfügbar sind. Computer-Techniker sind (speicher-) sparsame Leute.

Wenn als Tastatureingabe ein Wert gefunden wird, der größer als 127 ist, dann haben wir es mit einem negativen Zeichen zu tun. Wir ziehen den Wert 128 ab, um an den positiven Zwilling heranzukommen. Damit aber die Ausgabe auf dem Bildschirm in unserem Programm auch negativ aussieht, vertauschen wir die Farbwerte für Grafikpunkt und Hintergrund und das Zeichen wird negativ erscheinen.

Weil ein einmal gePOKEter Wert nicht so intelligent ist, selbst zu merken, wann er nicht mehr erwünscht ist, müssen wir in Zeile 240 dafür sorgen, daß die normale Zuordnung von Grafik und Hintergrundfarbe hergestellt wird, wenn das aufgerufene Zeichen aus der ersten Hälfte des Zeichensates (kleiner 128) stammt, klar doch.

GRAPHICS 1 und 2 - numerische Werte für COLOR											
Zeichen		Farbregister				Zeichen		Farbregister			
1.	2.	708	709	710	711	1.	2.	708	709	710	711
	♥	32	0	160	128	@	♦	64	96	192	224
!	├	33	1	161	129	A	a	65	97	193	225
"	│	34	2	162	130	B	b	66	98	194	226
#	┘	35	3	163	131	C	c	67	99	195	227
$	┤	36	4	164	132	D	d	68	100	196	228
%	┐	37	5	165	133	E	e	69	101	197	229
&	╱	38	6	166	134	F	f	70	102	198	230
'	╲	39	7	167	135	G	g	71	103	199	231
(	◢	40	8	168	136	H	h	72	104	200	232
)	▝	41	9	169	137	I	i	73	105	201	233
*	◣	42	10	170	138	J	j	74	106	202	234
+	▘	43	11	171	139	K	k	75	107	203	235
,	▗	44	12	172	140	L	l	76	108	204	236
-	▔	45	13	173	141	M	m	77	109	205	237
.	▁	46	14	174	142	N	n	78	110	206	238
/	▖	47	15	175	143	O	o	79	111	207	239
0	♣	48	16	176	144	P	p	80	112	208	240
1	┌	49	17	177	145	Q	q	81	113	209	241
2	─	50	18	178	146	R	r	82	114	210	242
3	┼	51	19	179	147	S	s	83	115	211	243
4	●	52	20	180	148	T	t	84	116	212	244
5	▬	53	21	181	149	U	u	85	117	213	245
6	▌	54	22	182	150	V	v	86	118	214	246
7	┬	55	23	183	151	W	w	87	119	215	247
8	┴	56	24	184	152	X	x	88	120	216	248
9	▐	57	25	185	153	Y	y	89	121	217	249
:	└	58	26	186	154	Z	z	90	122	218	250
;	♠	59	27	187	155	[	♠	91	123	219	251
<	↑	60	28	188	156	\	│	92	124	220	252
=	↓	61	29	189	157	]	↖	93	125	221	253
>	←	62	30	190	158	^	◀	94	126	222	254
?	→	63	31	191	159	_	▶	95	127	223	255

Die Werte n von 0 bis 255 bewirken in GR.1 und GR.2 als Befehl COLOR n das Zeichen in der entsprechenden Zeile und beziehen ihren Farbwert aus dem Register der entsprechenden Spalte. Die Werte 125 und 155 bewirken keine Ausgabe von Zeichen. Die Zeichen in der 2. Spalte können nur nach POKE 756,226 aufgerufen werden. POKE 756,224 (Grundwert) stellt auf den 1. Teil zurück.

Zeichen - ATASCII-Code - interner Code

♥	0	64	(	40	8	P	80	48
├	1	65	)	41	9	Q	81	49
│	2	66	*	42	10	R	82	50
┘	3	67	+	43	11	S	83	51
┤	4	68	,	44	12	T	84	52
┐	5	69	-	45	13	U	85	53
╱	6	70	.	46	14	V	86	54
╲	7	71	/	47	15	W	87	55
◢	8	72	0	48	16	X	88	56
▝	9	73	1	49	17	Y	89	57
◤	10	74	2	50	18	Z	90	58
▘	11	75	3	51	19	[	91	59
▗	12	76	4	52	20	\	92	60
▔	13	77	5	53	21	]	93	61
▁	14	78	6	54	22	^	94	62
▖	15	79	7	55	23	_	95	63
♣	16	80	8	56	24	♦	96	96
┌	17	81	9	57	25	a	97	97
─	18	82	:	58	26	b	98	98
┼	19	83	;	59	27	c	99	99
●	20	84	<	60	28	d	100	100
▬	21	85	=	61	29	e	101	101
│	22	86	>	62	30	f	102	102
┬	23	87	?	63	31	g	103	103
┴	24	88	@	64	32	h	104	104
▌	25	89	A	65	33	i	105	105
└	26	90	B	66	34	j	106	106
▐	27	91	C	67	35	k	107	107
↑	28	92	D	68	36	l	108	108
↓	29	93	E	69	37	m	109	109
←	30	94	F	70	38	n	110	110
→	31	95	G	71	39	o	111	111
	32	0	H	72	40	p	112	112
!	33	1	I	73	41	q	113	113
"	34	2	J	74	42	r	114	114
#	35	3	K	75	43	s	115	115
$	36	4	L	76	44	t	116	116
%	37	5	M	77	45	u	117	117
&	38	6	N	78	46	v	118	118
'	39	7	O	79	47	w	119	119

x	120	120
y	121	121
z	122	122
↑	123	123
|	124	124
↖	125	125
◀	126	126
▶	127	127
	128	192
	129	193
	130	194
	131	195
	132	196
	133	197
	134	198
	135	199
	136	200
	137	201
	138	202
	139	203
	140	204
	141	205
	142	206
	143	207
	144	208
	145	209
	146	210
	147	211
	148	212
	149	213
	150	214
	151	215
	152	216
	153	217
	154	218
	155	219
	156	220
	157	221
	158	222
	159	223
	160	128
	161	129
	162	130
	163	131
	164	132
	165	133

	166	134
	167	135
	168	136
	169	137
	170	138
	171	139
	172	140
	173	141
	174	142
	175	143
0	176	144
1	177	145
2	178	146
3	179	147
4	180	148
5	181	149
6	182	150
7	183	151
8	184	152
9	185	153
	186	154
	187	155
	188	156
	189	157
	190	158
	191	159
	192	160
	193	161
B	194	162
C	195	163
D	196	164
E	197	165
F	198	166
G	199	167
H	200	168
I	201	169
J	202	170
K	203	171
L	204	172
M	205	173
N	206	174
O	207	175
P	208	176
Q	209	177
R	210	178
S	211	179

T	212	180
U	213	181
V	214	182
W	215	183
X	216	184
Y	217	185
Z	218	186
[	219	187
\	220	188
]	221	189
^	222	190
_	223	191
◆	224	224
a	225	225
b	226	226
c	227	227
d	228	228
e	229	229
f	230	230
g	231	231
h	232	232
i	233	233
j	234	234
k	235	235
l	236	236
m	237	237
n	238	238
o	239	239
p	240	240
q	241	241
r	242	242
s	243	243
t	244	244
u	245	245
v	246	246
w	247	247
x	248	248
y	249	249
z	250	250
	251	251
	252	252
	253	253
	254	254
	255	255

ABS	ABS(n) findet den absoluten Wert von n (entfernt das Minuszeichen) (198)
ADR	ADR(n$) ermittelt die Speicheradresse in der n$ beginnt
AND	Boolescher Operator (129f)
ASC	ASC(n$) oder ASC("n") ermittelt den ATASCII-Wert von n$ oder das erste Zeichen aus "n" (100)
ATN	ATN(n) errechnet den Arcustangens
BYE	(B.) / zum Verlassen von BASIC
CHR$	CHR$(n) ermittelt das ATASCII-Zeichen mit dem Wert n (42, 95, 100f)
CLOAD	(CLOA.) / Laden eines Programms von Cassette
CLOG	CLOG(n) berechnet den dekadischen Logarithmus von n
CLOSE	(CL.) / CLOSE#n schließt den Datenkanal n (159)
CLR	CLR zum löschen aller Variablen (22)
COLOR	(C.) oder (CO.) / COLOR n ruft einen Farbton auf, der im zugeordneten Farbregister definiert ist (97)
COM	wie DIM
CONT	(CON.) setzt ein unterbrochenes Programm fort
COS	COS(n) errechnet den Cosinus von n (146ff)
CSAVE	(CS.) / zum Speichern eines Programms auf Cassette
DATA	(D.) / markiert eine Datenzeile (216f)
DEG	(DE.) / schaltet auf Grad (147)
DIM	(DI.) / DIM v(n) / DIM v(n,m) / DIM v$(n) reserviert Speicherplatz für indizierte Variablen (18f, 20ff)
DOS	(DO.) / ruft das Diskettenbetriebssystem auf
DRAWTO	(DR.) / DRAWTO n,m zieht eine Linie von einer zuvor gePLOTteten Stelle x,y zu der definierten Position n,m (48)
END	beendet das Programm
ENTER	(E.) / ENTER"k:filename.ext" liest das Programm filespec.ext von der Peripherie "k:" (14)

EXP	EXP(n) berechnet die Potenz zur Zahl e, die n ergibt.
FOR	(F.) / FOR v=n (TO m STEP p) bestimmt den Anfangswert der Variablen v als Schleifenzähler (33f, 123)
FRE	FRE(0) stellt den freien Speicherplatz fest (11)
GET	(GE.) / GET#n,v liest aus dem Datenkanal n ein Byte in die Variable v
GOSUB	(GOS.) / GOSUB n springt in das Unterprogramm, das bei Zeile n beginnt (30f)
GOTO	(G.) / GOTO n stetzt das Programm bei Zeile n fort (17, 28f)
GRAPHICS	(GR.) / GRAPHICS n ruft eine Grafikbetriebsart n (0 bis 15) auf, n+16 ohne Textfenster, n+32 ohne Bildschirmlöschen, n+48 ohne Textfenster und Bildschirmlöschen (36, 64ff)
IF	IF (Bedingung), bedingte Anweisung (THEN) (32f)
INPUT	(I.) / INPUT n oder INPUT n$ weist der Variablen n oder n$ eine Benutzer-Eingabe zu
INT	INT(n) rundet n, Dezimalstellen fallen weg (15, 99)
LEN	LEN(n$) stellt die Anzahl der Zeichen von n$ fest (26)
LET	(LE.) / LET v=n oder LET v$="n" weist einer Variablen einen Wert zu (16ff)
LIST	(L.) / LIST oder LIST"k:filename.ext"schreibt das Programm filename.ext an die Peripherie "k:". Zur Ausgabe auf den Bildschirm nur LIST. LIST oder LIST n oder LIST n,m schreibt das ganze Programm, schreibt nur die Zeile n, schreibt von Zeile n bis m. Auch: LIST "k:filename.ext",n,m (14, 24, 52ff, 104)
LOAD	(LO.) / LOAD"k:filesname.ext" lädt ein Programm von der Peripherie "k:" in den Computer
LOCATE	(LOC.) / LOCATE m,n,v findet an der Position n,m des Bildschirms einen ATASCII-Wert oder COLOR-Wert und liest ihn in v (123f)
LOG	LOG(n) findet den natürlichen Logarithmus von n
LPRINT	(LP.) / LPRINT "n" oder LPRINT CHR$(n) gibt "n" oder CHR$(n) an den Drucker aus

NEW	löscht ein Programm im Computer
NEXT	(N.) / NEXT v überwacht den Schleifenzähler v (33f, 123)
NOT	Boolescher Operator (129f)
NOTE	(NO.) / NOTE#n,s,b notiert über den Datenkanal n den Sektor s und das Byte b von dem ab das nächste Byte auf die Diskette geschrieben oder gelesen wird
ON	ON (Bedingung) definiert eine Bedingung für den GOTO- oder GOSUB-Sprung. ON n bestimmt als Sprungziel für GOTO oder GOSUB die n-te angegebebe Zeilennummer (28f, 30f)
OPEN	(O.) / OPEN#n,p,q,"k:filename.ext" eröffnet den Datenkanal n mit der Betriebsart p und der Kennzahl q zur Peripherie "k:" (159)
OR	Boolescher Operator (129f)
PADDLE	PADDLE(n) stellt die Stellung des Drehreglers n (0 bis 7) fest
PEEK	PEEK(n) liest den Inhalt der Speicherzelle n (23)
PLOT	(PL.) / PLOT n,m zeichnet einen Punkt nach n,m (47f)
POINT	(P.) / POINT#n,s,b bestimmt, das über den Datenkanal n im Sektor s das Byte b der Diskette gelesen oder beschrieben wird
POKE	(POK.) / POKE n,m schreibt in die Speicherzelle n den Wert m (23, 37)
POP	zum Verlassen eines Unterprogramms ohne RETURN (31)
POSITION	(POS.) / POSITION n,m stellt den unsichtbaren Cursor auf die Bildschirmstelle n,m (43ff, 47)
PRINT	(PR.)oder(?) / PRINT n oder PRINT n$ oder PRINT "n" gibt n, n$ oder "n" an den Bildschirm aus. Ein Datenkanal kann bestimmt werden: PRINT=n (16, 17, 24, 47)
PTRIG	PTRIG(n) stellt fest, ob der Druckknopf am Drehregler n (0 bis7) gedrückt ist
PUT	(PU.) / PUT#n,m gibt über den Datenkanal n den Wert m an die Diskette aus
RAD	schaltet auf Bogenmaß um (147)

READ	(REA.) / READ n oder READ n$ liest aus DATA-Zeilen in n oder n$ (216f)
REM	(R.)oder(.) nachfolgende Zeichen bis zum Ende der Zeile werden vom Computer nicht bearbeitet
RESTORE	(RES.) / setzt den READ-DATA-Zeiger auf die erste oder mit RESTORE n auf die n-te DATA-Zeile (217)
RETURN	(RET.) / führt aus einem Unterprogramm an die Stelle, wo das Hauptprogramm verlassen wurde
RND	RND(n) findet einen zufälligen Wert von gleich 0 bis kleiner 1 unabhängig von n (99)
RUN	(RU.) / beginnt die Abarbeitung eines Programms (16)
SAVE	(S.) / SAVE"k:filename.ext" speichert das Programm filename.ext auf die Periherie "k:"
SETCOLOR	(SE.) / SETCOLOR r,f,h definiert für das Farbregister r den Farbwert f*16+h (37f)
SGN	SGN(n) findet das Signum von n in der Form 1, 0 oder -1 (198)
SIN	SIN(n) berechnet den Sinus von n (146ff)
SOUND	(SO.) / SOUND r,t,k,v setzt in den Tongenerator r die Tonhöhe t mit der Verzerrung k und der Lautstärke v (212ff)
SQR	SQR(n) berechnet die Quadratwurzel von n
STATUS	(ST.) / STATUS#n,m liest über den Datenkanal n die Kennziffer des aufgetretenen Fehlers in m ein
STEP	STEP p bestimmt für den Schleifenzähler einer FOR-NEXT-Schleife die Schrittweite p (34)
STICK	STICK(n) stellt die Stellung des Steuerknüppels n (0 bis 3) fest (160ff)
STRIG	STRIG(n) stellt fest ob der Feuerknopf am Steuerknüppel n gedrückt ist (160ff)
STOP	programmierte Programmunterbrechung
STR$	STR$(n) wandelt die Zahl n in den String"n"um
THEN	THEN (Kommando) bestimmt, was nach der Bedingung (IF) geschehen soll. Das Kommando kann eine Zeilennummer (ohne GOTO) sein (32)
TO	TO n bestimmt den Endwert des Schleifenzählers,

dessen Anfangswert mit FOR v=n definiert wurde (33f, 123)

TRAP (T.) / TRAP n bestimmt die Zeile n, zu der nach Auftreten eines Fehlers gesprungen werden soll, statt das Programm mit einer Fehlermeldung zu unterbrechen (89, 216f)

USR USR(n) ruft ein in Maschinensprache geschriebenes Unterprogramm auf, das mit Speicherzelle n beginnt

VAL VAL(n$) macht aus der ersten Ziffer, die in n$ enthalten ist einen numerischen Wert

XIO (X.) / XIO n,#m,p,q,"k:filename.ext" ermöglicht Ein- und Ausgabeoperationen. n bestimmt die Art der Operation, m den Datenkanal, p die Betriebsart, q die Kennzahl und "k:" die Peripherie (50f)

Operationen, bei denen ein Gerätekennzeichen ("k:") angegeben werden muß, verlangen nur die Angabe des Dateinamens (filename.ext), wenn die Peripherie in der Lage ist, verschiedene Dateien zu verwalten (z.B. die Diskettenstation). Bei den übrigen Peripherien (Cassettenrecorder, Grafikfenster) kann der Dateiname fortgelassen werden.

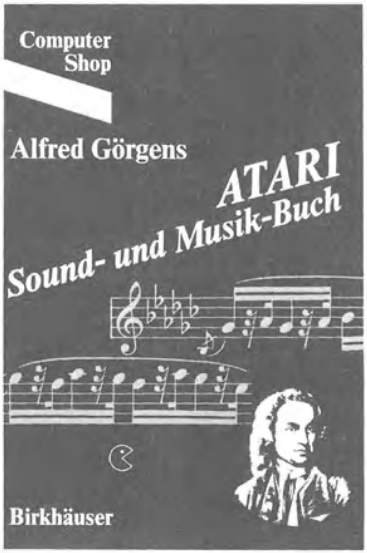
Birkhäuser Computer Shop

Alfred Görgens

ATARI Sound- und Musik-Buch

1984. 128 Seiten, Broschur.
ISBN 3–7643–1658–6

Soundeffekte machen Computerspiele perfekt. Aber wie soll man aus Hunderten von Frequenzen und sieben Verzerrungsgraden den «richtigen» Sound für bestimmte Programme finden? Das unterhaltsam geschriebene Buch vermittelt für Anfänger und Fortgeschrittene leicht verständlich, wie Töne und Effekte aus allen Programmbereichen erzielt werden können. In einem zweiten Teil beschreibt der Autor, wie Sie Ihren Computer in eine Orgel verwandeln und Rhythmuseffekte erzeugen können. Insgesamt ein grossartiges Buch für alle, die sich für Sound und Musik interessieren und vom einfachen BASIC bis zur Direktprogrammierung ihren ATARI-Computer beherrschen wollen.



Springer Basel AG